UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

Módulo de detección de vulnerabilidades en redes de computadoras para el sistema XILEMA GRHS

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

Hery Emanuel Fuka Rodrigues

Tutores:

MSc. Madelis Pérez Gil

Ing. Ramón Guzmán Alemañy

Cuba, La Habana, junio 2020

“Año 62 de la Revolución”

Declaración de autoría

Declaro ser el autor de la presente tesis que tiene por título: "***Módulo de detección de vulnerabilidades en redes de computadoras para el sistema XILEMA GRHS***", y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Hery Emanuel Fuka Rodrigues

Autor

_____ _____

Ing. Ramón Guzmán Alemañy MSc. Madelis Pérez Gil

Tutor Tutor

Datos de contacto

Tutor 1:

MSc. Madelis Pérez Gil: Ingeniera en Ciencias Informáticas 2008/ Universidad de las Ciencias Informáticas. Máster en Gestión de Información de la Facultad de Economía / Universidad de La Habana / Cátedra UNESCO 2014. Profesora Auxiliar del Dpto. de Ingeniería de Software de la Facultad 2. Vicedecana de Formación de la Facultad 2.

Email: mgil@uci.cu

Universidad de las Ciencia Informáticas, La Habana, Cuba.

Tutor 2:

Ing. Ramón Guzmán Alemañy: Ingeniero en Ciencias Informáticas 2017/ Universidad de las Ciencias Informáticas. Se desempeña como especialista A en Ciencias Informáticas en el Departamento de Desarrollo de Aplicaciones del Centro Telemática. Ha desempeñado el rol de desarrollador en el proyecto y hoy es el líder del proyecto XILEMA GRHS.

Email: ralemany@uci.cu

Universidad de las Ciencia Informáticas, La Habana, Cuba.

Agradecimientos

Hoy termina y cierra un capítulo en el libro de mi vida, uno que ha necesitado muchas horas de sacrificio y dedicación, y quería aprovechar este espacio para agradecer a todos los que hicieron posibles, que este sueño fuera realidad.

Primeramente, quería agradecer a Dios, por darme la vida y por permitir que llegue a este momento pese a muchos tropiezos y dificultades.

A toda mi familia y amigos que desde el primer día me han dado apoyo incondicional y desde lejos siempre me han enviado energías positivas, en especial a mi tío/papá ***Walter Nhumbuavali,*** *que es la persona que más apoyo incondicional me brinda en esta vida y que sabe que le amo mucho y a otro tío* ***Sebastião Fuka****, que tanto ha luchado junto a mí para que este sueño fuera realidad. Sin olvidarme de otras personas como Celmira, Wilson, Edmundo, Joel Navarro, Vornes, Tía Vania, Tío Guto, Pedro Chiveio, y Olga Francisco mi madrasta que es mi segunda madre y a mis hermanos y hermanas, mi mamá Ana Pereira que tantos consejos me dio y ayuda incondicional para que pueda cumplir mi sueño de hacerme ingeniero.*

Agradecimiento especial a mis tutores, ***Madelis Pérez Gil*** *y* ***Ramón Guzmán Alemañy*** *por su apoyo incondicional, por todos los consejos y enseñanzas que llevaré para la vida, que pese a esa situación difícil (covid-19), siempre han estado presentes para apoyarme y ayudarme en lo que me hace falta, en especial Madelis que sabe que le quiero muchísimo.*

Quiero agradecer la Universidad de las Ciencias Informáticas y la Facultad 1 por el apoyo que me han brindado en el momento más difícil de mi vida, cuando perdí a mi madre en tercer año. En especial a la profesora Delly Lien, Niurvis, Serguey, Daimara, Maikel, mis compañeros de aula y a todos los compañeros del edificio por todo su apoyo en ese momento tan difícil para mí. A mi mamacita cubana Daisy por todos sus consejos durante ese tiempo. A la profesora ***María Ramírez de la Caridad (****Rebeca) que desde la preparatoria aquí en Cuba me ha ayudado mucho, me ha enseñado el camino que debería seguir.*

A la Vicerrectora Natalia Martínez que si hoy estoy aquí es gracias a su regaño en la preparatoria, que me ha transformado en el estudiante que he sido durante ese tiempo, a usted en especial siempre voy a decir, esté donde esté, ¡MUCHAS GRACIAS!

Dedicatoria

Dedico mi tesis a mi mamá ***Angélica Fuka*** *que ya no se encuentra en este mundo y que, desde el inmenso cielo, sé que está orgullosa de que yo esté logrando su sueño.*

A mi abuela ***Deolinda Fuka*** *que también ya no se encuentra en este mundo, pero que junto a mi mamá hicieron de todo para que fuera mejor persona y que lograra realizar el sueño que hoy se ha concretado.*

Siempre estarán en mi corazón y sé que estén donde estén siempre van apoyarme y protegerme, a ustedes va dedica esta tesis….

Resumen

El inventario de los recursos de hardware y software en redes de computadoras es cada vez más, una tarea necesaria con la creciente evolución tecnológica. Con el fin de automatizar este difícil y complejo proceso se han creado innumerables herramientas informáticas en todo el mundo. El Centro de Telemática (TLM) de la Universidad de las Ciencias Informáticas (UCI) desarrolló una herramienta similar, denominada Gestor de Recursos de Hardware y Software (XILEMA GRHS), con el objetivo de inventariar los componentes de hardware y software en una red de computadoras. Sin embargo, XILEMA GRHS no permite descubrir los puertos abiertos en dichas computadoras inventariadas y qué vulnerabilidades representan para las redes de computadoras, lo cual constituye una deficiencia del sistema que influye negativamente en el inventario del hardware y el software de las redes de computadoras donde esté desplegado XILEMA GRHS. En la presente investigación se implementó un módulo que garantiza la detección de las vulnerabilidades generadas por los puertos abiertos para el sistema XILEMA GRHS. Para guiar el desarrollo de la misma se utilizó como metodología de desarrollo de software, el Proceso Unificado Ágil (AUP) para la UCI, como lenguaje de programación Python, HTML y JavaScript, como marco de trabajo Django, y como entorno de desarrollo integrado (IDE) el PyCharm

Palabras Clave: Masscan, Nmap, Puertos abiertos, Red de computadoras, Vulnerabilidades, XILEMA GRHS.

Abstract

Inventory of hardware and software resources for computer networks is increasingly a necessary task with increasing technological evolution. In order to automate this difficult and complex process, countless computer tools have been created around the world. The Telematics Center (TLM) of the University of Computer Sciences (UCI) developed a similar tool, called the Hardware and Software Resource Manager (XILEMA GRHS), with the aim of inventorying the hardware and software components for network computers. However, XILEMA GRHS does not allow discovering the open ports on these inventoried computers and what vulnerabilities they represent for computer networks, which constitutes a deficiency of the system that negatively influences the inventory of the hardware and software for computer networks where XILEMA GRHS is deployed. In the present investigation, a module was implemented that guarantees the detection of the vulnerabilities generated by open ports for the XILEMA GRHS. To guide the development of results management, the Agile Unified Process (AUP) for the UCI was used as a software development methodology, as a Python, HTML and JavaScript programming language, as a Django framework, and as an integrated development environment (IDE) the Pycharm.

Keywords: *Computer Network, Masscan, Nmap, Open Ports, Vulnerabilities, XILEMA GRHS.*

Índice

Índice de figuras

Índice de tablas

Introducción

El uso de las Tecnologías de la Información y las Comunicaciones (TIC) y del internet, ha significado a escala mundial un salto acelerado en el desarrollo científico técnico. De todos los elementos que integran las TIC, el que más ha impactado en la sociedad es Internet (Castells, 2013).

Desde la consolidación de internet como medio de interconexión global, la cantidad de ataques contra sistemas computacionales y los servicios que se ejecutan se ha incrementado de manera alarmante, pues los piratas informáticos avanzaron a pasos agigantados desarrollando técnicas y métodos para vulnerar los sistemas de seguridad. Ese hecho, unido a la progresiva dependencia de las mayoría de las organizaciones hacia sus sistemas de información, viene provocando que las mismas implementen mecanismos de prevención que reduzcan al mínimo los riegos de seguridad asociados a vulnerabilidades generadas por los puertos abiertos en redes de computadoras, que pueden ser : contraseñas débiles, robo de información, perdida y manipulación de datos, interrupción de servicios que se están ejecutando entre otras (Javier, 2015).

Cuba, con el objetivo de lograr un desarrollo tecnológico, contribuir al fortalecimiento de la soberanía tecnológica y al desarrollo del software libre, ha desarrollado un programa para la informatización de la sociedad. Programa en el cual se encuentra inmersa la Universidad de las Ciencias Informáticas (UCI); donde a partir del vínculo docencia - investigación - producción como modelo de formación, ejerce de soporte a la industria cubana del software. La UCI cuenta con una amplia red de centros de desarrollo de software, que realizan proyectos informáticos, con gran impacto en la informatización de la sociedad cubana. Entre ellos se encuentra el centro de Telemática (TLM), que tiene como objetivo, desarrollar sistemas y servicios informáticos integrales para las Telecomunicaciones y la Seguridad Informática, altamente comprometidos con la Revolución, capaz de integrar los procesos docentes, productivos e investigativos con alto nivel, contando con un personal especializado en

dichas áreas” . En función de garantizar el uso correcto por parte de los usuarios a los servicios de red, la UCI posee su propio reglamento de seguridad informática, donde se definieron políticas de seguridad para todos los usuarios, algunas como: No brindar servicios telemáticos sin autorización tales como HTTP, FTP, SMB, correo, jabber, tener instalado antivirus, tener el cliente de XILEMA GRHS, compartir carpetas con la debida protección entre otras.

El centro TLM cuenta con un sistema Gestor de Recursos de Hardware y Software, es una aplicación informática que tiene como objetivo: realizar el inventario de la información de los componentes de Hardware y Software en una red de computadoras. Dentro de las principales funcionalidades del mismo se encuentran:

- ✓ Facilitar la detección de los cambios en dichos componentes, clasificarlos en autorizados o no autorizados, según se haya establecido en la entidad.
- ✓ Tomar acciones de control (apagar, hibernar, suspender, reiniciar, enviar notificación por correo) ante los cambios no autorizados.

A pesar de las características citadas anteriormente y de las ventajas que suponen las mismas, XILEMA GRHS posee la siguiente limitación:

- ✓ El sistema no realiza la detección automática de los puertos abiertos (puertos conocidos 1-1023) en las computadoras inventariadas, lo que puede provocar la pérdida de datos de las computadoras inventariadas o el acceso a la información de los inventarios por parte de las personas no autorizadas.

Teniendo en cuenta la situación problemática anterior se plantea como **problema a resolver**: Las limitaciones que presenta XILEMA GRHS en la detección de vulnerabilidades, influye negativamente en el control del sistema y puede comprometer la integridad de los datos que se obtienen cuando se realiza un inventario.

Introducción

Se define como **objeto de estudio** el proceso de detección de vulnerabilidades en redes informáticas. Estableciéndose como **objetivo general** del presente trabajo: Desarrollar un módulo para la detección de las vulnerabilidades en redes de computadoras mediante XILEMA GRHS.

Definiéndose como **campo de acción** el proceso de detección de vulnerabilidades en los puertos, para las redes de computadoras que utilizan XILEMA GRHS.

Para alcanzar el objetivo trazado y dar solución al problema planteado se elaboraron las siguientes **tareas de investigación**:

1. Revisión de la bibliografía para elaborar el marco teórico conceptual en lo referente a aplicaciones que realizan escaneo de vulnerabilidades generadas por los puertos abiertos.
2. Realización del estado del arte analizando los sistemas similares que permiten detectar las vulnerabilidades generadas por los puertos abiertos en redes de computadoras.
3. Identificación de las herramientas y lenguajes informáticos, además de la metodología de desarrollo de software asumida por el proyecto XILEMA GRHS para realizar la implementación del sistema.
4. Identificación de las principales funcionalidades del sistema a desarrollar para la posterior implementación del mismo, teniendo en cuenta los requisitos definidos anteriormente.
5. Ejecución de los diferentes tipos de pruebas de software, para evitar posibles errores en el sistema desarrollado.

Las **preguntas científicas** que guían y orientan el desarrollo del proceso investigativo son las siguientes:

- ¿Cuáles son los referentes teóricos a tener en cuenta para elaborar el marco conceptual en lo referente a aplicaciones que realizan escaneo de vulnerabilidades generadas por los puertos abiertos?

- ¿Cómo identificar los sistemas similares que permiten detectar vulnerabilidades generadas por los puertos abiertos en redes de computadoras?
- ¿Cuáles son las herramientas y lenguajes informáticos, además de la metodología de desarrollo de software asumido por el proyecto XILEMA GRHS?
- ¿Cuáles son las principales funcionalidades del sistema a desarrollar, teniendo en cuenta los requisitos definidos anteriormente?
- ¿Cómo se valida el correcto funcionamiento del módulo para la detección de vulnerabilidades generadas por los puertos abiertos en las redes de computadoras que utilizan XILEMA GRHS?

Para el desarrollo de la investigación se utilizaron diferentes métodos científicos de investigación que sirvieron para comprender la realidad del estudio de la naturaleza, la sociedad y el pensamiento, con el fin de manifestar su esencia y sus relaciones. Dichos métodos se clasifican en teóricos y empíricos, dentro de los **métodos teóricos** utilizados en la investigación se encuentran:

Analítico-sintético: Este método sirvió para la recopilación de información requerida durante la realización del estado del arte y para el desarrollo del trabajo mediante la revisión de documentos y artículos, de donde se extrajeron los elementos más significativos relacionados con el escaneo de vulnerabilidades generados por puertos abiertos. Además del análisis de las diferentes herramientas, metodología y tecnologías a utilizar en el desarrollo del módulo.

Histórico-Lógico: Este método permitió el análisis de la evolución de sistemas similares, de esta manera se profundizó sobre los rasgos que caracterizan a estos sistemas y en los aspectos principales para fundamentar la propuesta de solución a la problemática planteada.

Modelación: Fue utilizado en la representación, mediante el uso de diagramas, de las características del sistema a desarrollar, relaciones entre objetos; y las actividades que

intervinieron en el proceso de detección de vulnerabilidades generado por los puertos abiertos en redes informáticas.

El método empírico utilizado fue:

Entrevista: Este método se utilizó para el desarrollo del módulo, detectando las necesidades del cliente a través del intercambio con él, para identificar los principales problemas existentes y así poder satisfacer sus necesidades y obtener un software con la calidad requerida, que permitió contar con la información necesaria, convirtiéndose en una técnica de recopilación. Se realizan una serie de entrevistas específicamente a especialistas del proyecto XILEMA GRHS, para poder entender mejor la arquitectura modelo-vista-controlador y los patrones de diseño a usar. Dicha entrevista se encuentra en el Anexo 1.

El documento está estructurado por 3 capítulos, las conclusiones generales, las referencias bibliográficas, utilizadas y los anexos. La estructura de los capítulos se define a continuación:

Capítulo 1: **Fundamentación teórica**. En este capítulo se abordan los principales presupuestos teóricos asociados a la investigación, se realiza un estudio de algunos sistemas que poseen un comportamiento identifico al módulo a desarrollar, y además se describen la metodología, las herramientas informáticas y lenguajes a utilizar para el desarrollo del módulo.

Capítulo 2. Análisis y diseño de la solución: En este capítulo se realiza el análisis y diseño del módulo de detección de vulnerabilidades en redes de computadoras para el sistema XILEMA GRHS. Se presenta la propuesta de solución al problema planteado, quedando definidos los requisitos funcionales y no funcionales del sistema. Así como todos los artefactos que propone la metodología de desarrollo de software. También se detalla la arquitectura del sistema y elementos del diseño del sistema.

Capítulo 3: Implementación y pruebas del sistema: Se describen los estándares de codificación seguidos para la implementación de la propuesta de solución. Así como representación del diagrama despliegue correspondiente con dicha propuesta. Para culminar se representan las pruebas realizadas al sistema, así como el resultado de las pruebas realizadas.

Capítulo I. Fundamentación teórica

1.1- Introducción

En el presente capítulo se abordan los se realiza el análisis de los diferentes sistemas que realicen acciones similares a las que se quieren desarrollar en el módulo en cuestión. También se caracteriza la metodología, las tecnologías, lenguajes y herramientas informáticas a utilizar para el desarrollo de la solución.

1.2- Estado del Arte

Una de las primeras etapas que debe desarrollarse dentro de una investigación es el estado del arte ya que permite determinar la forma como ha sido tratado el tema, así como el estudio de las principales aplicaciones utilizadas actualmente en el mundo y en Cuba que puedan contribuir para el desarrollo del módulo a desarrollar. El estado del arte hace referencia a la construcción de un análisis de tipo documental, eso permite evitar duplicar esfuerzos o repetir lo que ya se ha dicho. Esta muestra los avances más importantes que se han logrado con respecto al conocimiento de un tema.

1.2.1- Internacional

- **Nessus**

El proyecto Nessus fue iniciado y liberado bajo licencia General Public Licence (GPL, por sus siglas en inglés) en el año de 1998 por Renaud Deraison, es un programa de escaneo de vulnerabilidades en diversos sistemas operativos. Consiste en un proceso, llamado "*Nessusd"*, que realiza el escaneo en el sistema objetivo, y *Nessus*, el cliente (basado en consola o gráfico) que muestra el avance e informa sobre el estado de los escaneos. En operación normal, *Nessus* comienza escaneando los puertos con Nmap o con su propio escaneador de puertos para buscar puertos abiertos y después

intentar varios exploits para atacarlo. Las pruebas de vulnerabilidad, disponibles como una larga lista de plugins, son escritos en **NASL** (*Nessus Attack Scripting Language*, Lenguaje de Scripting de Ataque Nessus por sus siglas en inglés), un lenguaje scripting optimizado para interacciones personalizadas en redes. Opcionalmente, los resultados del escaneo pueden ser exportados como informes en varios formatos, como texto plano, XML, HTML, y LaTeX. Los resultados también pueden ser guardados en una base de conocimiento para referencia en futuros escaneos de vulnerabilidades (Beale et al. , 2016).

Vale enfatizar que desde el año de 2005, la empresa Tenable ha comprado la licencia de Nessus, pasando así a ser una herramienta de pago (Tenable, 2015) .

- **Nmap**

Nmap fue desarrollado en el año 1997 y liberado bajo licencia General Public Licence (GPL, por sus siglas en inglés) en septiembre de 1997 por Gordon Lyon. Fue publicado por primera vez en la revista *Phrack Magazine*, localizada en St. Louis, Estados Unidos de América. Es una herramienta de análisis en cuanto a seguridad. Permite obtener prácticamente todo tipo de información sobre los equipos de una red. Esta herramienta es capaz de detectar *hosts*, puertos, servicios activos, tipo de sistema operativo, *firewall* y aplicaciones que están utilizando la red. También es capaz También es capaz de realizar escaneos de forma oculta (indetectable para *firewalls*). Realiza escaneos sobre ciertos puertos, entre rangos de IP (Lyon, 2017).

- **Masscan**

Masscan fue desarrollado en el año de 2013 en la Universidad de Míchigan, Estados Unidos y liberado bajo licencia General Public Licence (GPL, por sus siglas en inglés) en octubre de 2014 por Robert Graham. Es un escáner de puertos, es compatible

principalmente para sistemas operativos Linux, pero también lo es con *Windows, Mac OS X* y *FreeBSD*. Según pruebas realizadas por los autores, en *Windows* y *Mac OS X* se puede enviar hasta 300.000 paquetes por segundo, y en *Linux* llega hasta 1,5 millones de paquetes por segundo. El uso de esta herramienta se realiza a base de comandos, muy parecidos a Nmap. Masscan proporciona la posibilidad de obtener el *banner* desde los puertos identificados en estado abierto. Esta herramienta se considera familiar para los usuarios de Nmap, aunque sean fundamentalmente diferentes. Existen dos importantes diferencias: no existen puertos por defectos a escanear, siempre se debe especificar y los *hosts* objetivos deber ser siempre direcciones a través del protocolo de internet (IP, por sus siglas en inglés) o rangos simples (Quezada, 2017).

1.2.2- Nacional-UCI

- **Shodan-UCI**

Desarrollada en el año de 2017 en la UCI en el año de 2017, liberada hasta la fecha como software libre, es un sistema para la detección automática de publicación de servicios telemáticos no autorizados en la red de la UCI, tiene como objetivo principal la detección de puertos y de definir las violaciones que constituyen para la red de la UCI (Guzmán y Rodríguez, 2017) .

1.2.3- Bibliotecas utilizadas para detección de puertos abiertos en la red

A continuación, se describen las principales bibliotecas que serán utilizadas para la detección de puertos abiertos en la red.

- **Python-Nmap:** Es una biblioteca de Python que ayuda a usar el escáner de puertos Nmap. Permite manipular fácilmente los resultados del escaneo de

Nmap y es bastante útil para administradores de sistemas que desean automatizar escaneos e informes (pypi.org, 2016).

- **Python-Massca:** Es una biblioteca de Python que ayuda a usar el escáner de puertos Masscan. Devuelve el resultado del escaneo similar a Nmap, con la diferencia de que es escáner de puertos más rápido que hay actualmente en el mundo (pypi.org, 2019).

1.3- Conclusiones del estado de arte

En la tabla 1 se hace un resumen de algunas herramientas informáticas estudiadas y sus principales características principales, marcando con una X las características que poseen dichos sistemas. Esas son:

- **Descubrimiento de servidores:** Se refiere a la capacidad del sistema de descubrir los servidores instalados en la red.
- **Puertos abiertos:** Se refiere a la detección de puertos y las vulnerabilidades asociadas a los mismos.
- **Servicios que se están ejecutando:** Se refiere a los servicios que se están utilizando (FTP, SMB, HTTP entre otros).
- **Sistema Operativo:** Se refiere al descubrimiento de los Sistema Operativos que están utilizando todas las computadoras en la red.
- **Escaneo programado:** Se refiere a la capacidad de realizar escaneo de forma programada de formas a que no congestione la red.
- **Captura web:** Es un recurso que utiliza fotografiar las violaciones cometidas por los usuarios y guardarlas como evidencia.

Tabla 1: Resumen del estado del arte

Características	*Nmap*	*Masscan*	*Nessus*	*Shodan-UCI*
Descubrimiento de servidores	*X*			
Puertos abiertos	*X*	*X*	*X*	*X*
Servicios que se están ejecutando	*X*		*X*	
Sistema Operativo	*X*	*X*		
Escaneo programado	*X*		*X*	*X*
Captura Web		*X*		*X*

El estudio realizado arrojó los siguientes resultados:

1. Solamente el Nmap realiza descubrimientos de servidores en la red.

2. Todas las herramientas realizan descubrimiento de puertos abiertos en la red.

3. Solamente Nmap y Nessus son capaces de detectar los servicios que se están ejecutando en el momento.

4. Apenas Nessus y Masscan detectan el sistema operativo que utilizan las computadoras en la red.

5. El único sistema que no realiza escaneo programado es Masscan.

6. Shodan-UCI y Masscan son los únicos que realizan captura web.

Todas las herramientas descritas tienen como propósito fundamental realizar escaneo y detección de puertos abiertos en una red de computadoras, cada una con sus características específicas. Se tomaron las características antes expuestas porque son las funcionalidades que actualmente presentan los sistemas de escaneo de vulnerabilidades generadas por los puertos abiertos y algunas de estas fueron especificadas por el cliente para el desarrollo del módulo. Además, se seleccionaron las bibliotecas Python-Nmap y Python-Masscan para la implementación del sistema, haciendo una combinación entre la rapidez de Python-Masscan y la eficiencia de Python-Masscan.

1.4- Metodología de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. Las técnicas indican cómo debe ser realizada una actividad técnica determinada identificada en la metodología. Combina el empleo de unos modelos o representaciones gráficas junto con el empleo de unos procedimientos detallados (Maida y Pacienzia, 2015).

- **AUP versión para la UCI**

El Proceso Unificado Ágil de Scott Amblero Agile Unified Process (AUP) en inglés es una versión simplificada del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que se llamarán Ejecución y se agrega una fase de Cierre (Sánchez, 2015).

AUP-UCI propone para el ciclo de vida de los proyectos en la UCI tener siete disciplinas, los flujos de trabajo son los siguientes*: Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas, Pruebas de liberación y Pruebas de aceptación.* Además, está compuesta por cuatro escenarios:

- **Escenario I:** Proyectos que modelen el negocio solo pueden modelar el Sistema con Caso de Uso de Sistema (CUS).
- **Escenario II**: Proyectos que modelen el negocio con Modelo Conceptual (MC), solo pueden modelar el sistema con Casos de Uso del Sistema (CUS)

- **Escenario III**: Proyectos que modelen el negocio con Descripción de Proceso de Negocio (DPN), solo pueden modelar el sistema mediante la Descripción de Requisitos por Proceso (DRP).

- **Escenario IV**: Proyectos que no modelen el negocio, solo pueden modelar el sistema con Historias de Usuario (HU).

Para la presente investigación, se trabaja sobre el escenario 2. El mismo se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, porque el sistema es el responsable por la gestión de la información de forma automática, de esa forma modelan exclusivamente los conceptos fundamentales del negocio. También se toma este escenario pues como política del centro de desarrollo TLM es el escenario con el que se trabaja y todos sus productos están enmarcados en el mismo.

1.5- Lenguajes, herramientas y tecnologías informáticas

Un lenguaje de programación es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador poder expresar el procesamiento de datos y sus estructuras en la computadora (Porto y Merino , 2016). Para el desarrollo del módulo de detección de vulnerabilidades en una red de computadoras serán utilizadas las mismas tecnologías que utiliza el sistema XILEMA GRHS, empleando como lenguaje del lado del servidor Python 2.7 y como lenguajes del lado del cliente HTML5 y JavaScript. A continuación, se muestra una breve descripción de cada uno de ellos.

1.5.1- Lenguaje informático del lado del servidor

- **Python 2. 7**

Es un lenguaje de programación multipropósito de alto nivel, simple y sencillo de aprender además es libre. Se propone su utilización pues brinda muchas funcionalidades para aplicaciones web que trabajen a un nivel bien cercano al sistema operativo, además posee mecanismos eficientes para su integración con bases de datos (Python.org, 2019). Se decidió utilizar Python porque el sistema XILEMA GRHS está implementado sobre este lenguaje, además el componente a implementar se va a integrar a este sistema y debe ser desarrollado bajo sus mismas tecnologías. En el centro de TLM se definió este lenguaje de programación para sus productos, lo que beneficia que cualquier persona pueda darle mantenimiento al sistema o programar nuevas funcionalidades cuando sea necesario.

1.5.2 - Lenguaje informático del lado del cliente

- **HTML5**

Es la última evolución de la norma que define el lenguaje de marcas de hipertexto (HTML por sus siglas en inglés). Se trata de una nueva versión del lenguaje HTML, con nuevos elementos, atributos y comportamientos. Diseñado para ser utilizado por todo tipo de desarrolladores. Describe la estructura y el contenido semántico de un documento web. Se utilizará el lenguaje HTML5 para el diseño de las interfaces del componente, siendo este el utilizado en el sistema XILEMA GRHS actualmente (González y Rodríguez, 2017).

- **JavaScript**

Es un lenguaje de programación interpretado que se utiliza principalmente para crear páginas web dinámicas. JavaScript, es un lenguaje de script multiplataforma basado en objetos. Es un lenguaje ligero, está diseñado para una fácil incrustación en otros

productos y aplicaciones, tales como los navegadores web. Se utiliza para construir los gráficos de cada módulo del sistema XILEMA GRHS (González y Rodríguez, 2017).

1.5.3- Marco de trabajo

Un marco de trabajo se puede definir como un conjunto de componentes que estructuran un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web. Se definieron como marcos de trabajo a utilizar en el desarrollo del módulo de detección de vulnerabilidades en una red mediante XILEMA GRHS a Django 1.8 (García-Peñalvo, 2019).

- **Django 1.8**

Es un marco de trabajo (framework, por sus siglas en inglés) de desarrollo web sobre Python que permite desarrollar rápidamente aplicaciones web. Define una forma de desarrollar software que cuenta con 3 capas donde el Modelo es la capa de acceso a los datos, la plantilla es capa de presentación y la vista es la capa lógica del negocio. Se centra en automatizar todo lo posible (García, 2015). Es el marco de trabajo que se utilizará para la implementación del componente por ser actualmente el que se emplea en el sistema XILEMA GRHS.

1.5.4- Herramienta informática utilizada:

- **Visual Paradigm**

Es una herramienta CASE: (Ingeniería de Software Asistida por Computación por sus siglas en español). La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación (Pressman, 2002).

1.5.5- Entorno de desarrollo

Un Entorno Integrado de Desarrollo (IDE de su significado en inglés Integrated Development Environment), es un conjunto de herramientas que ha sido creado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Un IDE provee un marco de trabajo amigable para la mayoría de los lenguajes de programación y en algunos casos puede funcionar como un sistema en tiempo de ejecución en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto (CYTA, 2018).

- **PyCharm 2019**

Es un IDE o entorno de desarrollo integrado multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Proporciona soporte para el desarrollo web con Django.PyCharm es desarrollado por la empresa JetBrains y debido a la naturaleza de sus licencias tiene dos versiones, la *Community* que es gratuita y orientada a la educación y al desarrollo puro en Python y la *Professional*, que incluye más características como el soporte a desarrollo web (JetBrains, 2017). Los desarrolladores utilizan al Pycharm para el desarrollo del sistema XILEMA GRHS.

1.5.6- Sistema Gestor de Base de Datos

Un Sistema Gestor de Base de Datos **(**SGBD**)** o DataBase Managenent System (DBMS en inglés) es un sistema que permite la creación, gestión y administración de bases de datos**,** así como la elección y manejo de las estructuras necesarias para el almacenamiento y búsqueda de información del modo más eficiente posible (Marín, 2019).

- **PostgreSQL 9.5**

Es un potente sistema de base de datos relacional de objetos de código abierto que usa y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de manera segura las cargas de trabajo de datos más complicadas. PostgreSQL se ha ganado una fuerte reputación por su arquitectura comprobada, confiabilidad, integridad de datos, conjunto de características robustas, extensibilidad y la dedicación de la comunidad de código abierto detrás del software para ofrecer soluciones innovadoras y de alto rendimiento. Además de todas las características antes mencionadas, es el que utiliza el sistema XILEMA GRHS en la actualidad (PostgreSQL, 2018).

1.6- Conclusiones parciales

En el presente capítulo mediante un estudio del estado del arte se analizaron los principales elementos que se consideran importantes para sustentar la propuesta de solución y se llega a las siguientes conclusiones, el estudio del estado del arte con el objetivo de identificar tendencias actuales relacionadas con las funcionalidades, permitió identificar las características principales de los escáneres de vulnerabilidades relacionada con los puertos abiertos. El estudio de las tecnologías, herramientas, lenguajes y metodologías permite un mejor desempeño en el desarrollo de la solución.

Capitulo II. Análisis y diseño de la solución

2.1- Introducción

En el presente capítulo se diseñan los artefactos que propone la metodología de desarrollo de software seleccionada en el capítulo anterior. Para la realización de los artefactos fue necesaria la captura de los requisitos del software y a partir de estos se describen los diagramas de clases de diseño y los diagramas de secuencia. También se expone la arquitectura del sistema, los patrones de diseño que fueron utilizados y el modelo de diseño de la aplicación.

2.2- Características de la propuesta de solución

La propuesta de solución está enfocada a desarrollar un módulo de detección de vulnerabilidades para el sistema XILEMA GRHS. Realizando un escaneo en la red el cual permitirá obtener la información referente a los puertos abiertos y las vulnerabilidades que esos representan. Las vulnerabilidades encontradas se almacenan en una Base de Datos que solo puede ser accedida por los administradores del sistema. El sistema permitirá conocer el nombre de las computadoras, el IP, el Sistema Operativo que utilizan, el número del puerto y su estado. En la figura 1, se muestra las tareas que debe realizar e administrador para realizar un escaneo.

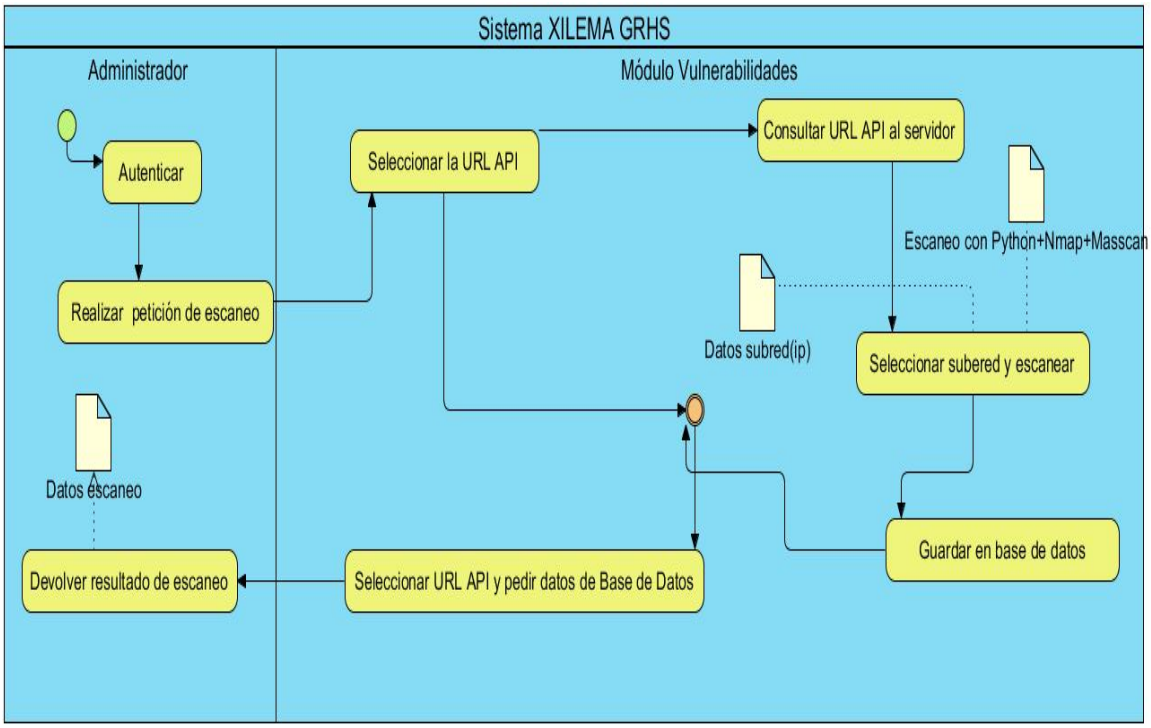

Figura 1: Propuesta de solución

2.3- Modelo conceptual

El modelo conceptual es una representación que se realiza para entender el proyecto donde se está trabajando. Sirve para identificar las relaciones que contienen todas las entidades de un sistema y contiene conceptos que estarán asociados tanto a su definición natural como al papel que juegan desde el punto de vista informático (Infante, 2013). Como se observa en la figura 2.

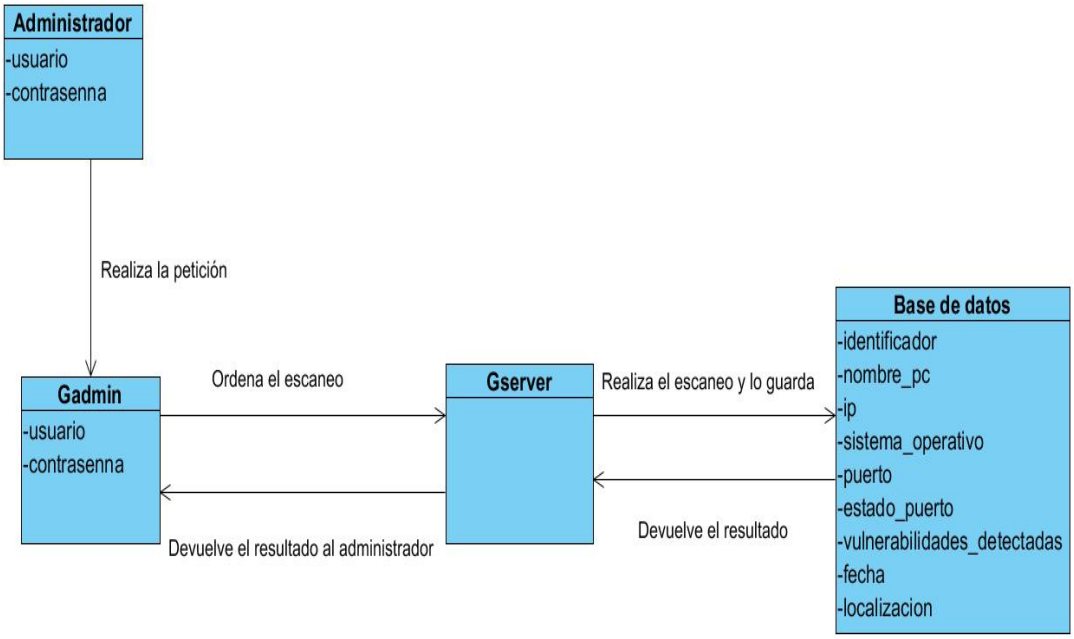

Figura 2: Modelo Conceptual

Se crean 4 entidades (Administrador, Gadmin, Gserver, Base de datos), todas están relacionadas entre sí orientadas a describir la semántica y observaciones sobre la información del dominio que en este caso es el escaneo de las vulnerabilidades generadas por los puertos abiertos. La clase administrador es donde están los datos del mismo, que se autentica al sistema a través de la interfaz de usuario Gadmin. La entidad Gserver es el servidor que provee el resultado del escaneo y ordena que se guarde en la base de datos, pudiendo ser solicitado los datos en cualquier momento que necesite el administrador.

2.4.- Especificación de los requisitos de software

Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan la necesidad de los clientes de un sistema que ayude a resolver algún determinado problema como el

control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar esos servicios y restricciones es denominado ingeniería de requisitos (Sommerville, 2007).

2.4.1- Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Eses requisitos dependen del sistema del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por el cliente al redactar los requisitos (Somerville, 2006).

A continuación, muestra las funcionalidades del software:

RF1- Ejecutar detección de vulnerabilidades.

RF2- Listar vulnerabilidades detectadas.

RF3- Exportar listado de vulnerabilidades detectadas en formato Excel.

RF4- Buscar en las vulnerabilidades detectadas según criterio:

- Por el identificador
- Por el Nombre de la PC
- Por el IP
- Por el Sistema Operativo
- Por el puerto
- Por el estado del puerto
- Por vulnerabilidades detectadas
- Por fecha
- Por localización de la PC

2.4.2- Requisitos no funcionales

Los requisitos no funcionales, como su nombre sugieren, son aquellos que no se refieren directamente a las funcionalidades específicas que proporciona el sistema, sino a las propiedades de éste (Somerville, 2006). Tales como:

Confiabilidad

- RNF. El sistema debe ser capaz de recuperarse ante fallos.
- RNF. El sistema debe ser capaz de prevenir el acceso no autorizado a los datos.

Usabilidad

- RNF. El sistema debe ser fácil de entender para los usuarios.
- RNF. El sistema debe realizar todas las operaciones solicitadas por el usuario.

Hardware

- RNF. Definir la cantidad de memoria RAM, capacidad de disco duro a utilizar en los clientes, el servidor y el servidor de bases de datos.

Portabilidad

- RNF. El sistema debe ser adaptable al sistema operativo Windows o GNU/Linux.
- RNF. El sistema debe poderse instalar en el sistema operativo Windows o GNU/Linux.

El proceso productivo de la UCI está certificado por el proceso de integración de sistemas modelos de madurez de capacidades (CMMI, por sus siglas en inglés) en el nivel 2 e implementan las plantillas que se describen en dicho proceso para los requisitos no funcionales.

En las siguientes tablas, se describen los requisitos no funcionales tomados del expediente del proyecto XILEMA GRHS, agrupados por sus atributos de calidad:

Tabla 2 : Requisito no funcional de Confiabilidad (Tolerancia a fallos)

Atributo de Calidad	*Confiabilidad*
Sub-atributos/Sub-características	*Tolerancia a fallos*
Objetivo	*Lograr que el sistema sea capaz de recuperarse ante fallos.*
Origen	*Interno al sistema*
Artefacto	*Servicios del sistema (cliente y servidor) / Canales de comunicación*
Entorno	*Operación normal*
Estímulo	***Respuesta: Flujo de eventos (Escenarios)***
1.a Interrupción de comunicaciones de red en la PC	
	Tratar de conectarse cada cierto tiempo para terminar el proceso de comunicación / Continuar

	funcionando en modo normal
Medida de respuesta	
NA	
2.a Interrupción de comunicaciones de red en la PC servidor	
	Tratar de conectarse cada cierto tiempo para terminar el proceso de comunicación / Continuar funcionando en modo normal
Medida de respuesta	
NA	
3.a Inactividad del cliente	
	El servidor se encargará de notificar los clientes inactivos / Continuar funcionando en modo normal
Medida de respuesta	
NA	
4.a Ocurrencia de una excepción	
	Se notifica al usuario / Continuar funcionando en modo normal
Medida de respuesta	
NA	

Tabla 3 : Requisito no funcional de Confiabilidad (Acceso no autorizado)

Atributo de Calidad	*Confiabilidad*
Sub-atributos/Sub-características	*Cumplimento de fiabilidad*
Objetivo	*Prevenir el acceso no autorizado, a los datos.*
Origen	*El Usuario*
Artefacto	*Canal de Comunicación, Sistema*
Entorno	*El usuario desea obtener información sobre los datos procesados por el sistema.*
Estímulo	*Respuesta: Flujo de eventos (Escenarios)*
1. a Persona no autorizada intenta acceder a los datos del sistema.	
	Bloquear el acceso al sistema
Medida de respuesta	
NA	
2. a Persona con permisos restringidos intenta acceder a información a la que no tiene acceso.	
	Bloquear el acceso a estos datos.
Medida de respuesta	
NA	

Tabla 4 : Requisito no funcional de Usabilidad (Comprensibilidad)

Atributo de Calidad	*Usabilidad*
Sub-atributos/Sub-características	*Comprensibilidad*
Objetivo	*Lograr que el sistema sea entendible fácilmente para los usuarios.*
Origen	*El usuario*
Artefacto	*El sistema*
Entorno	*El sistema está funcionando correctamente.*
Estímulo	***Respuesta: Flujo de eventos (Escenarios)***
NA	*NA*
Medida de respuesta	
NA	

Tabla 5 : Requisito no funcional de Usabilidad (Operabilidad)

Atributo de Calidad	*Usabilidad*
Sub-atributos/Sub-características	*Operabilidad*
Objetivo	*Lograr que el sistema sea capaz de realizar todas las operaciones solicitadas por el cliente.*
Origen	*El usuario*

Artefacto	*El sistema*
Entorno	*El sistema está funcionando correctamente.*
Estímulo	***Respuesta: Flujo de eventos (Escenarios)***
NA	*NA*
Medida de respuesta	
NA	

Tabla 6: Requisito no funcional de Hardware (Utilización de recursos)

Atributo de Calidad	*Hardware*
Sub-atributos/Sub-características	*Utilización de recursos*
Objetivo	*CPU (4 x 2.33 GHz (Intel Xeon 5140 Core2 2.33 GHz))* *RAM (8GB)* *HDD (30 Gb RAID 5)* *LAN (1 x NIC (1 Gbit))*
Origen	*Externo al sistema*
Artefacto	*Servidor de aplicación*
Entorno	*Operación normal*

Estímulo	*Respuesta: Flujo de eventos (Escenarios)*
NA	*NA*

Tabla 7: Requisito no funcional de Hardware (Utilización de recursos)

Atributo de Calidad	*Hardware*
Sub-atributos/Sub-características	*Utilización de recursos*
Objetivo	*CPU (4 x 2.33 GHz (Intel Xeon 5140 Core2 2.33 GHz))* *RAM (8GB)* *HDD (50 Gb RAID 5)* *LAN (2 x NIC (1 Gbit))*
Origen	*Externo al sistema*
Artefacto	*Servidor de almacenamiento de información*
Entorno	*Operación normal*
Estímulo	*Respuesta: Flujo de eventos (Escenarios)*
NA	*NA*

Tabla 8 : Requisito no funcional de Hardware (Utilización de recursos)

Atributo de Calidad	*Portabilidad*

Sub-atributos/Sub-características	*Adaptabilidad*
Objetivo	*El sistema debe ser adaptable al sistema operativo Windows o GNU/Linux*.
Origen	*El usuario*
Artefacto	*El sistema*
Entorno	*Funcionando correctamente*
Estímulo	***Respuesta: Flujo de eventos (Escenarios)***
1. a Se prueba el sistema en diferentes entornos.	
	El sistema funciona correctamente.
Medida de respuesta	
NA	

Tabla 9 : Requisito no funcional de Portabilidad (Instabilidad)

Atributo de Calidad	*Portabilidad*
Sub-atributos/Sub-características	*Instabilidad*
Objetivo	*El sistema debe poderse instalar en el sistema operativo Windows o GNU/Linux.*
Origen	*El usuario*

Artefacto	*El sistema*
Entorno	*Funcionando correctamente*
Estímulo	***Respuesta: Flujo de eventos (Escenarios)***
1.a Se instala el sistema en diferentes entornos.	
	El sistema funciona correctamente.
Medida de respuesta	
NA	

2.5- Casos de uso del sistema

El diagrama de casos de uso representa la forma en como un cliente (actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso) (UNAD, 2020).

En la tabla 10, se describen los actores y en la figura 3 los casos de uso del sistema.

Tabla 10: Descripción de autores

Actor	***Descripción del actor***
Sistema XILEMA GRHS	*Ejecuta el escaneo para poder encontrar posibles vulnerabilidades.*
Administrador	*Gestiona el almacenamiento de la información referente a las vulnerabilidades encontradas.*

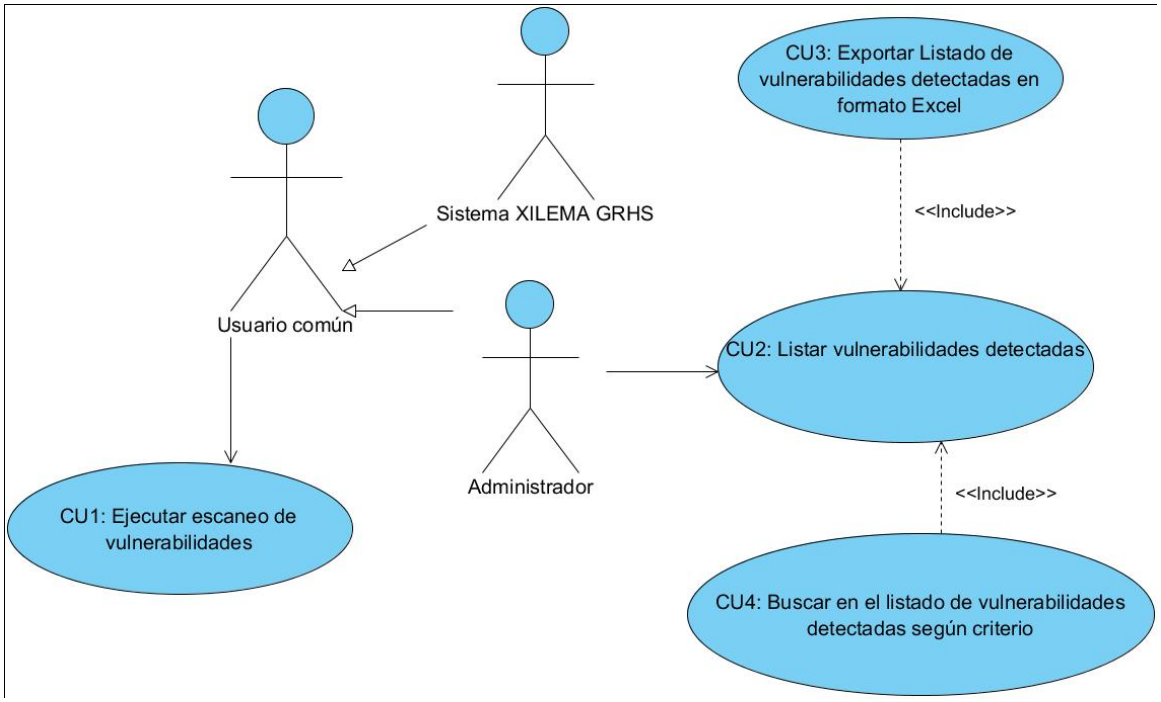

Figura 3: Diagrama de caso de uso

2.5.1- Descripción de los casos de uso

Para una mejor comprensión de las funcionalidades asociadas a cada caso de uso es necesario describirlos. Dicha descripción puede ser elaborada de forma breve o extendida. Una especificación de caso de uso proporciona detalles textuales de un caso de uso. En las siguientes tablas, se describen detalladamente los casos de usos encontrados:

Tabla 11 : Ejecutar escaneo de vulnerabilidades

Objetivo	*Escanear la red y encontrar posibles vulnerabilidades*	
Actores	*Sistema*	
Prioridad	*Alta*	
Precondiciones	*El usuario se ha autenticado en el sistema XILEMA GRHS y posee permisos de administrador del mismo. El Gserver de XILEMA GRHS debe estar instalado en al menos una máquina.*	
Postcondiciones	*Detectar y almacenar las vulnerabilidades encontradas.*	
Flujo de eventos		
Flujo básico: Ejecutar escaneo y detectar posibles vulnerabilidades.		
	Actor	**Sistema**
•	*El administrador selecciona la URL API Inventario, en seguida clica en Políticas y después en el módulo Vulnerabilidades.*	
•		*El sistema XILEMA GRHS inicializa el servicio Gserver.*
		Ejecuta el escaneo en la red que esté instalado el Gserver a través de Python, Nmap y Masscan.
•		*Almacena los datos obtenidos en la base de datos.*
•		*Termina el caso de uso.*
Flujos alternos		

1. **No existe conexión con la base de datos.**		
	Actor	**Sistema**
1.		*El sistema XILEMA GRHS obtiene la información del escaneo y comprueba la conexión con la base de datos, hasta que exista conexión entre los dos.*
Asuntos Pendientes: NA		
Prototipo de interfaz gráfica de usuario		
		

Tabla 12 : Listar vulnerabilidades detectadas

Objetivo	*Listar las vulnerabilidades encontradas*
Actores	*Administrador*
Prioridad	*Alta*
Precondiciones	*El usuario ha sido autenticado en el sistema XILEMA GRHS y posee*

<table>
<tr><td></td><td colspan="2">permisos de administrador del mismo.</td></tr>
<tr><td>Postcondiciones</td><td colspan="2">Devolver las vulnerabilidades encontradas.</td></tr>
<tr><td colspan="3">Flujo de eventos</td></tr>
<tr><td colspan="3">Flujo básico: Listar las vulnerabilidades encontradas.</td></tr>
<tr><td></td><td>Actor</td><td>Sistema</td></tr>
<tr><td>•</td><td>El administrador selecciona la URL API Inventario, después Políticas y en seguida en el módulo Vulnerabilidades.</td><td></td></tr>
<tr><td></td><td></td><td>El sistema XILEMA GRHS devuelve los datos de las vulnerabilidades encontradas con los siguientes campos:
Identificador
Nombre de la PC
IP
Sistema Operativo
Puerto
Estado
Vulnerabilidades detectadas
Fecha
Localización</td></tr>
<tr><td>•</td><td>El administrador obtiene el listado con todos los campos.</td><td></td></tr>
<tr><td>•</td><td></td><td>Termina el caso de uso</td></tr>
<tr><td colspan="3">Flujos alternos</td></tr>
<tr><td colspan="3">1. No fue detectada ninguna vulnerabilidad.</td></tr>
<tr><td></td><td>Actor</td><td>Sistema</td></tr>
</table>

1.		*El sistema XILEMA GRHS notifica al administrador de que no fue detectada ninguna vulnerabilidad.*
Asuntos Pendientes : NA		
Prototipo de interfaz gráfica de usuario		

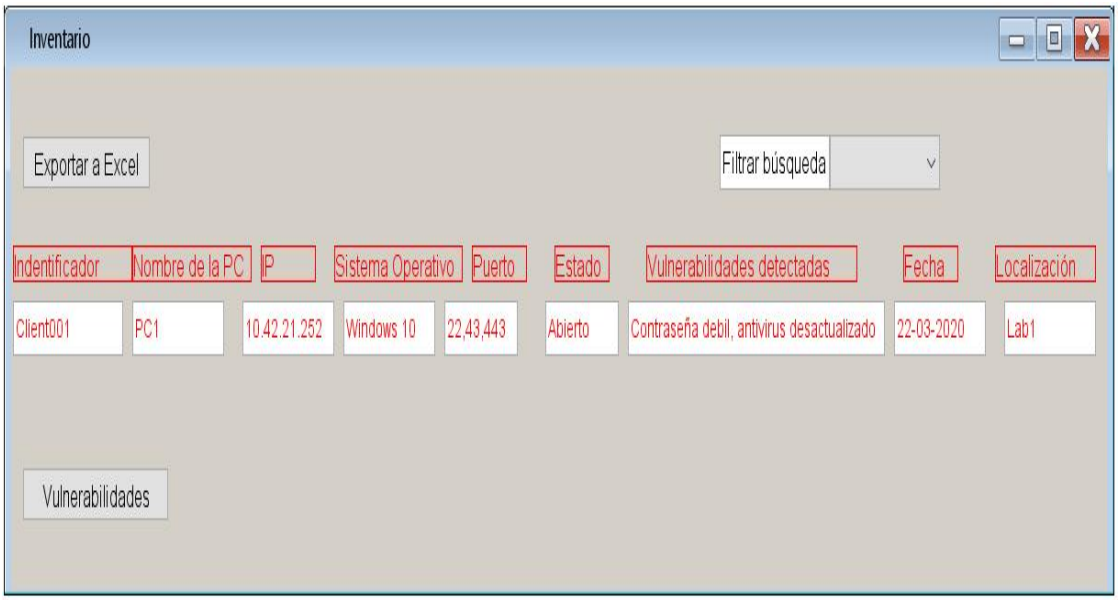

Tabla 13 : Exportar vulnerabilidades detectadas en formato Excel

Objetivo	*Exportar listado de vulnerabilidades detectadas en formato Excel*
Actores	*Administrador*
Prioridad	*Media*
Precondiciones	*El usuario ha sido autenticado en el sistema XILEMA GRHS y posee permisos de administrador del mismo.*
Postcondiciones	*Almacenar toda la información de la base de datos referente a las*

	vulnerabilidades encontradas en un fichero de formato Excel.
Flujo de eventos	
Flujo básico: *Exportar listado de vulnerabilidades detectadas en formato Excel*.	

	Actor	Sistema
•	*El administrador selecciona el icono en la parte superior izquierda "Exportar a Excel".*	
•		Verifica que el listado de vulnerabilidades está en la base de datos y exporta.
		Llena la planilla una en formato Excel con todos los datos encontrados.
•		*Termina el caso de uso.*

Flujos alternos

1. **No hay información del listado en la base de datos.**
2. **No existe conexión con la base de datos.**

	Actor	Sistema
1.		*El sistema XILEMA GRHS emite un mensaje de que no hay datos a exportar.*
2.		*El Sistema XILEMA GRHS obtiene la información del escaneo y comprueba la conexión con la base de datos, hasta que exista conexión entre los dos.*

Asuntos pendientes: NA	
Prototipo de interfaz de Usuario	

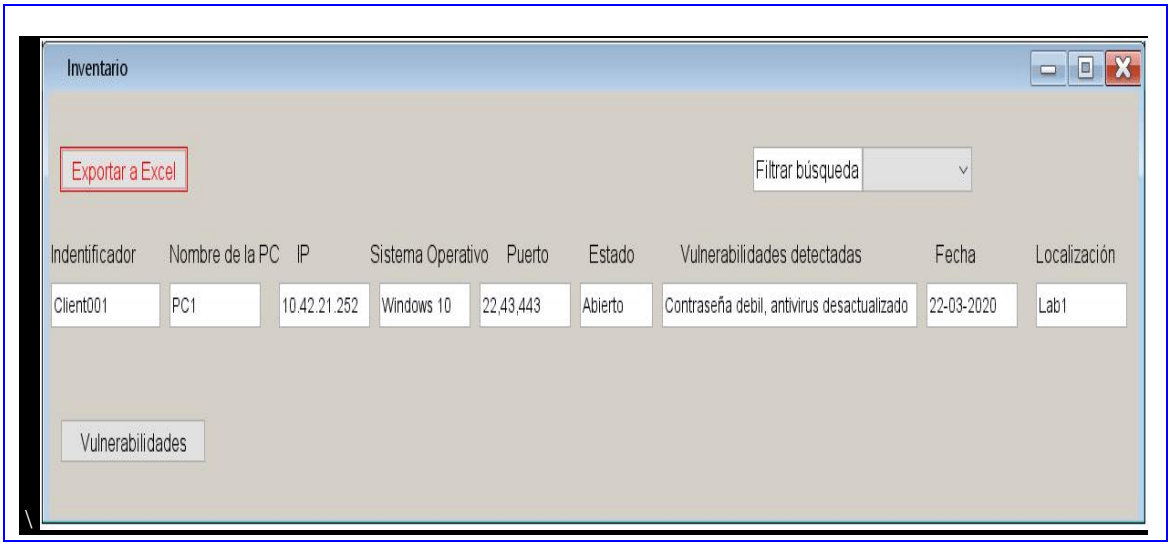

Tabla 14: Buscar en el listado de vulnerabilidades detectadas según algún criterio

Objetivo	*Buscar en el listado de las vulnerabilidades según algún criterio*
Actores	*Administrador*
Prioridad	*Alta*
Precondiciones	*El usuario ha sido autenticado en el sistema XILEMA GRHS y posee permisos de administrador del mismo.*
Postcondiciones	*Buscar en las vulnerabilidades según algún criterio*
Flujo de eventos	
Flujo básico: Listar criterios de búsqueda.	

	Actor	Sistema
•	*El administrador selecciona "Filtrar búsqueda".*	

•		*El sistema XILEMA GRHS muestra las opciones de búsqueda, aparece una ventana con el criterio escogido, esos son:* • *Identificador* • *Nombre de la PC* • *IP* • *Sistema Operativo* • *Puerto* • *Estado del puerto* • *Vulnerabilidades detectadas.* • *Fecha* • *Localización de la PC*
	Aparece una ventana con el criterio seleccionado y el administrador, introduce el dato.	
		El sistema XILEMA GRHS devuelve el dato según el criterio escogido por el administrador.
		Termina el caso de uso.
Asuntos pendientes : NA		
Prototipo de interfaz de usuario		

2.6- Modelos de Datos

En el modelo de base de datos se determina la estructura lógica de una base de datos, el modo de almacenar, organizar y manipular los datos. Se utiliza para representar la base de datos de manera gráfica a través del diagrama de entidad relación. La figura 4, muestra las tablas empleadas en la confección del modelo de base de datos. Estas tablas almacenan los valores de importancia que van a ser obtenidos de cuando se realizan los escaneos.

Las tablas que se muestran a continuación son las que intervienen directamente en el sistema a desarrollar.

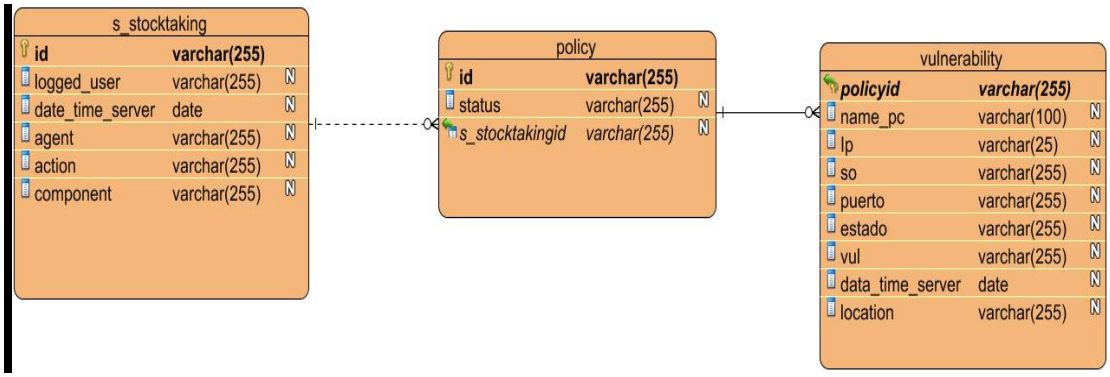

Figura 4: Modelo físico de datos

- **Descripción del modelo físico de base de datos**

En las siguientes tablas, se describen las tablas que componen al modelo físico de base de datos, la llave primaria se denomina PK (por sus siglas en inglés) y la llave foránea se denomina FK (por sus siglas en inglés).

Tabla 15: Descripción de la tabla s_stocktaking

Nombre: s_stocktaking				
Descripción: Contiene los datos de los módulos de seguridad.				
Nombre:	*Tipo de dato*	*PK/FK*	*¿Nulo?*	*Descripción:*
Id	*varchar*	*PK*	*No*	*Identifica con una cadena de caracteres a cada uno de los*

				módulos de seguridad.
looged_user	*date*		*No*	*Guarda el usuario que está accediendo al escaneo.*
agent	*varchar*	*FK*	*No*	*Almacena el identificador del agente que realizó el escaneo.*
date_time_server	*varchar*		*No*	*Guarda la fecha de la obtención de la información del escaneo.*
action	*varchar*		*No*	*Indica la acción realizada por el agente que ha realizado el escaneo.*
component	*varchar*		*No*	*Indica sobre cuáles componentes ha realizado una determinada acción.*

Tabla 16: Descripción de la tabla policy

Nombre: policy
Descripción: Contiene los datos de las políticas de seguridad

Nombre:	*Tipo de dato*	*PK/FK*	*¿Nulo?*	*Descripción:*
Id	*varchar*	*PK*	*No*	*Identifica con una cadena de caracteres a cada uno de los módulos de seguridad.*
status	*date*		*No*	*Identifica el estado de las computadoras (encendido, apagado).*
s_stocktakingid	*varchar*	*FK*	*No*	*Es la llave foránea, que es la llave primaria de la tabla s_stocktakingid*

Tabla 17: Descripción de la tabla vulnerability

Nombre: vulnerability				
Descripción: Contiene los datos de las vulnerabilidades.				
Nombre:	*Tipo de dato*	*PK/FK*	*¿Nulo?*	*Descripción:*
policyid	*varchar*	*PK*	*No*	*Hereda la llave de la tabla policy, porque es una entidad débil.*
name_pc	*varchar*		*No*	*Identifica el nombre de las computadoras escaneadas.*
Ip	*varchar*	*FK*	*No*	*Indica la dirección Ip de las computadoras escaneadas.*
so	*varchar*		*No*	*Indica el sistema operativo de las computadoras escaneadas.*
puerto	*varchar*		*No*	*Indica los números de los puertos abiertos detectados.*

estado	*varchar*		*No*	*Indica el estado de los puertos (abierto, cerrado, filtrado).*
vul	*varchar*		*No*	*Indica las vulnerabilidades generadas por los puertos abiertos.*
date_time_server	*date*		*No*	*Indica la fecha en que se ha realizado el escaneo.*
location	*varchar*		*No*	*Indica la ubicación física de la computadora (laboratorio, docente, edificio entre otras)*

2.7. Arquitectura del sistema

La arquitectura del software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores. Es definida según la IEEE Estándar 1471-2000 como: la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos, el contexto en el que se implantarán, y los principios que orientan su diseño y evolución (Fernández, 2006).

2.7.1- Patrones de arquitectura

Los patrones arquitectónicos se utilizan para expresar una estructura de organización base o esquema para un software. Proporcionando un conjunto de sub-sistemas predefinidos, especificando sus responsabilidades, reglas, directrices que determinan la organización, comunicación, interacción y relaciones entre ellos (ingeniods, 2013).

- **Modelo Cliente-Servidor**

El sistema XILEMA GRHS cuenta con una arquitectura cliente-servidor, el cual permite a los usuarios finales obtener acceso a la información de forma transparente y segura aún en entornos multiplataforma. En la arquitectura cliente servidor, el cliente (Gclient) envía un mensaje solicitando un determinado servicio al servidor (Gserver), o lo que es lo mismo le hace una petición, y este envía uno o varios mensajes con la respuesta (provee el servicio) a través de la red. Como se muestra en la figura 5.

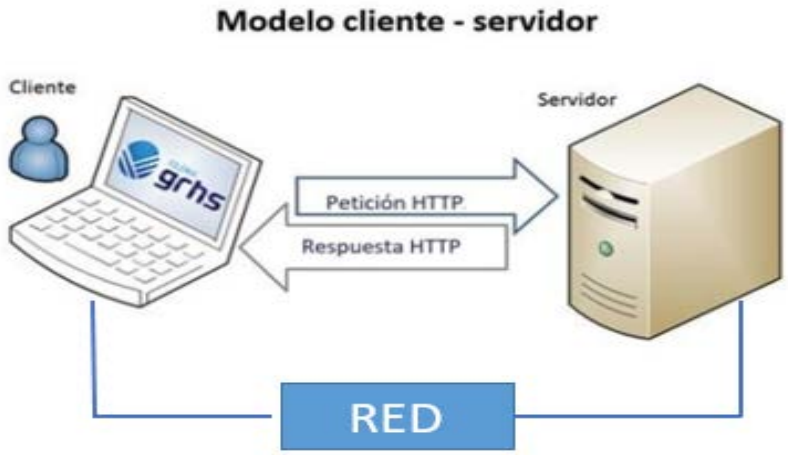

Figura 5: Arquitectura Cliente Servidor

Modelo Vista Plantilla (MTV): Django se basa en el Modelo Plantilla Vista (MPV), es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos (Django , 2015). Ese patrón se encuentra en lado del servidor de XILEMA GRHS (Gserver).

Modelo (Model): Es la parte de acceso a la base de datos, es manejada por la capa de la base de datos Django.

Plantilla (Template): Es la parte que contiene las decisiones relacionadas a la presentación, es el modo en el que algunas cosas son mostradas sobre una página web u otro tipo de documento y son los componentes que muestran la interfaz de usuario del sistema.

Vista (View): Es la parte lógica del negocio que contiene la lógica que accede al modelo y la relaciona con la plantilla apropiada y es el componente que administra y controla la interacción del usuario.

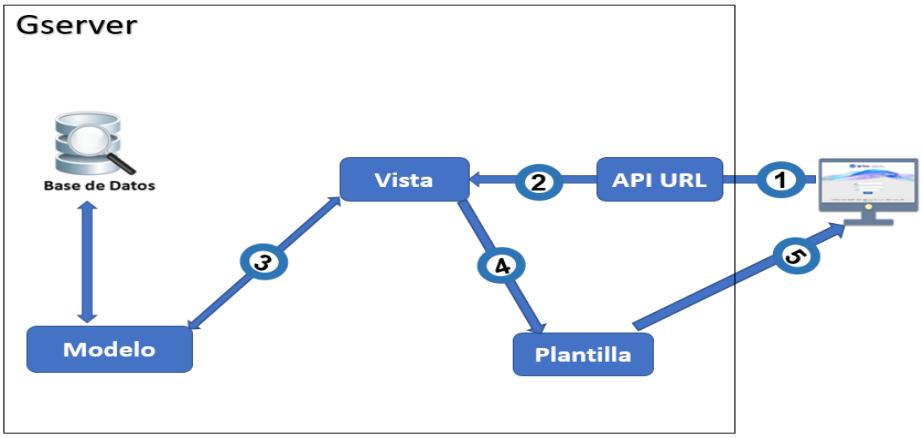

Figura 6: Patrón arquitectónico Modelo Vista Plantilla (Gserver)

Explicación del Modelo Vista Plantilla que ilustra la figura 6.

1- Primeramente, el navegador envía una solicitud (Gadmin) a través de una URL API seleccionada.
2- La URL API interpreta la solicitud y ubica la vista apropiada.
3- Seguidamente la vista se comunica con el modelo para obtener los datos.
4- La vista llama a la plantilla a través de la URL API seleccionada.

5- La plantilla responde a la solicitud del navegador (Gadmin), y muestra el resultado de la solicitud.

2.8- Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces (Luján-Mora y Valarezo, 2014).

Patrones GRASP

Los patrones GRASP (General Responsibility Assignment Software Patterns, por sus siglas en inglés) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable (Saavedra, 2007). A continuación, se explican los patrones GRASP que fueron utilizados para el desarrollo del módulo (Pressman, 2005).

- **Patrón creador:** Guía la asignación de responsabilidades y ayuda a identificar quién debe ser el responsable de la creación de objetos. Se asigna la tarea a una clase de crear cuando, contiene, agrega, compone, almacena o usa otra clase. El propósito fundamental de este patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. Este patrón se evidencia en la clase vulnerability_model.py.

- **Patrón experto:** Se usa al asignar responsabilidades. Ofrece como solución asignar las responsabilidades a las clases que tienen la información necesaria para cumplir con estas, para las cuales son creadas sin depender de ninguna

otra. Es un principio básico que suele utilizarse en el diseño orientado a objetos (Pressman, 2005). Este patrón se evidencia en la clase vulnerability_model.py
.

- **Patrón alta cohesión:** El patrón alta cohesión es la meta principal que ha de buscarse en todo momento, se debe tener presente en todas las decisiones de diseño. El alto nivel de cohesión es presentado por las clases que tienen responsabilidades moderadas en un área funcional y colaboran con otras para llevar a cabo las tareas, no donde dichas clases abarcan el volumen de las responsabilidades a realizar sin importar su complejidad. Este patrón se evidencia en la clase vulnerability.py.
- **Patrón bajo acoplamiento:** El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de éstas no incremente el acoplamiento, es decir, significa asignar una responsabilidad para mantener pocas dependencias entre las clases. Este patrón se evidencia en la clase scaner.py

Patrones GoF

Los patrones GOF presentan tres tipos de clasificaciones, las cuales son utilizadas en dependencia del propósito que se quiere alcanzar, estas son (Larman, 2003):

- **Patrones de Creación:** Tratan la creación de instancias, y se abstraen a la forma en que los objetos son creados, de manera que permite tratar las clases a implementarse de forma genérica, y sin considerar las clases que serán desarrolladas ni la forma en que se hará.

- **Patrones Estructurales:** Tratan la relación entre clases, la combinación clases y la formación de estructuras más complejas.

- **Patrones de Comportamiento:** Tratan la interacción y la cooperación entre clases u objetos.

Los patrones GoF utilizados en el desarrollo del módulo son los mismos que utilizan Django, que son los patrones de comportamiento que se describen a continuación:

- **Patrón de comando**

Encapsula una petición como un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma (Moran y Araujo, 2018). Ese patrón se evidencia en todo el sistema, porque cuando el usuario realiza cualquier solicitud, la vista devuelve un objeto del tipo HTTPRequest, y este encapsula la información de manera a que no se conozca el contenido interno del objeto que está siendo solicitado.

- **Patrón observador**

Define una dependencia de uno a muchos entre los objetos, de tal forma que cuando el estado de un objeto cambia, todos sus dependientes son notificados y actualizado automáticamente (Moran y Araujo, 2018). Ese patrón se evidencia en la clase upadater.py, que realiza una actualización interna en todos los módulos del sistema y en la base de datos cuando ocurre un evento tal como: eliminar, salvar, actualizar y crear.

- **Patrón método de plantilla**

Define el esqueleto de un algoritmo en una operación, delegando alguno de los pasos a sus subclases. El template method Pattern permite a las subclases redifinir algunos pasos de un algoritmo sin cambiar la estructura del algoritmo (Moran y Araujo, 2018) .

Ese patrón se evidencia en todo sistema con el uso de la vista generica (generic view, por sús siglas en ingles), dónde están configuradas todas las direcciones (URL, en inglés) internas del, permite que al seren configuradas de forma generica, si se redifne la misma, no cambiaria la estructura generica.

2.9- Modelo de diseño

Describe la manera de comunicarse el software dentro de sí mismo, con sistemas que interoperan dentro de él y con las personas que lo utilizan (Allamandri, 2012). El modelo de diseño se realizó haciendo uso de la herramienta CASE-Visual Paradigm donde se definió una estructura de paquetes para la organización del trabajo que se mostrará a continuación.

2.9.1 Diagrama de clases de diseño

Los diagramas de clases sirven para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenido (Tarazona y Gómez, 2016). Los diagramas también se pueden representar como notas, restricciones, paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes.

Las clases que se muestran en la figura 7 son las que intervienen directamente en el sistema a desarrollar.

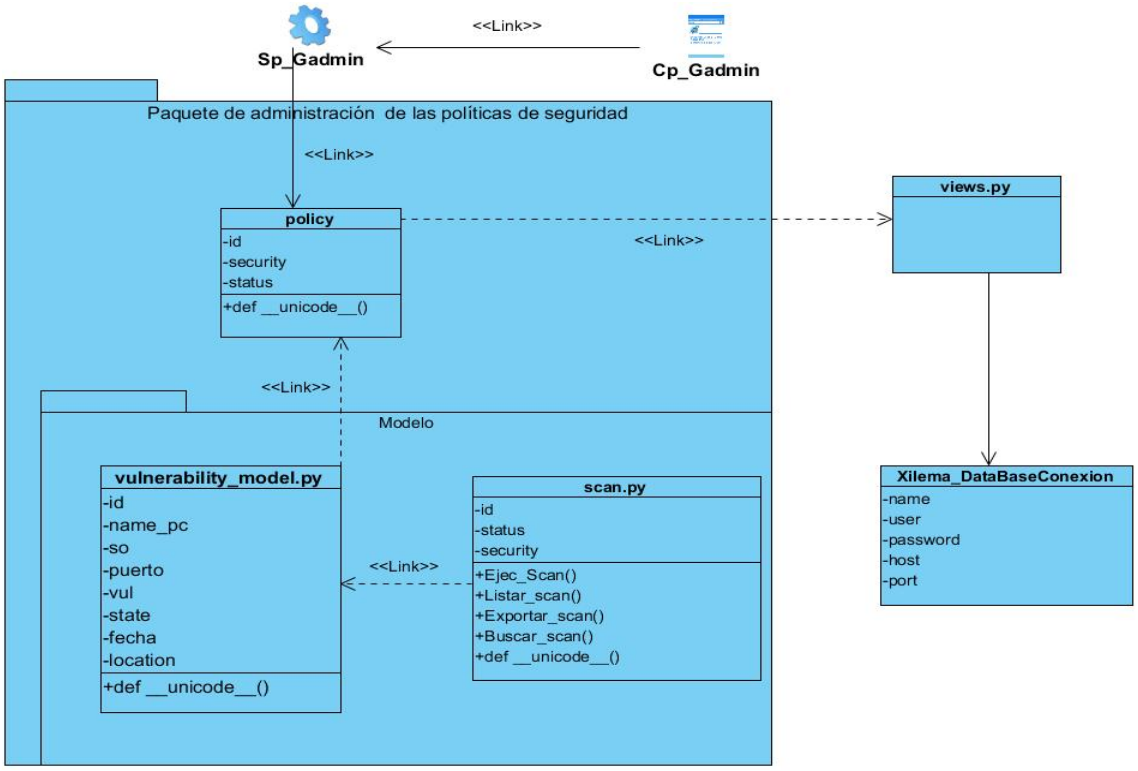

Figura 7: Diagrama de clases del diseño

Descripción de las clases

A continuación se describen las clases modeladas anteriormente:

- **policy:** Es el paquete que contiene las clases que componen las politicas de seguridad, es dónde se encuentra el paquete vulnerabily.
- **vulnerability_model.py:** Es el paquete que contiene la clase principal del negocio y el modelo del negocio.
- **scan.py:** Es la clase principal del negocio, contiene las funcionalidades principales referentes al escaneo de vulnerabilidades.

- **Xilema_Databaseconexion:** Es la clase que contiene los atributos necesarios para conectar la base de datos al sistema.
- **Views.py:** Contiene la lógica del proyecto.
- **Gserver (SP):** Representa la clase encargada de realizar el escaneo en busca de posibles vulnerabilidades.
- **Gadmin (CP):** Representa la página web encargada de mostrar la información del escaneo al usuario.

2.9.2 Diagrama de secuencia

El diagrama se secuencia muestra la interacción de un conjunto de objetos de una aplicación a través del tiempo, en el cual se indicarán los módulos o clases que formarán parte del programa y las llamadas que se hacen a cada uno de ellos para realizar una tarea determinada, por esta razón permite observar la perspectiva cronológica de las interacciones (Cevallos, 2015).

Los diagramas que se muestran a continuación representan las clases del diseño con las que se tuvo interacción durante la presente investigación:

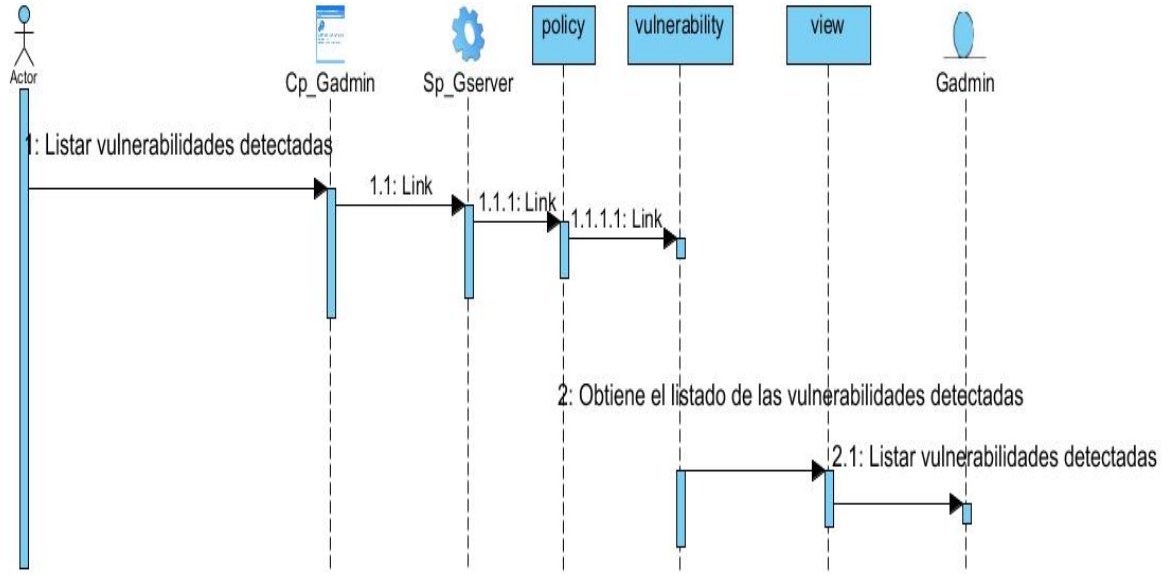

Figura 8: Diagrama de secuencia del Requisito Funcional (Listar Vulnerabilidades detectadas)

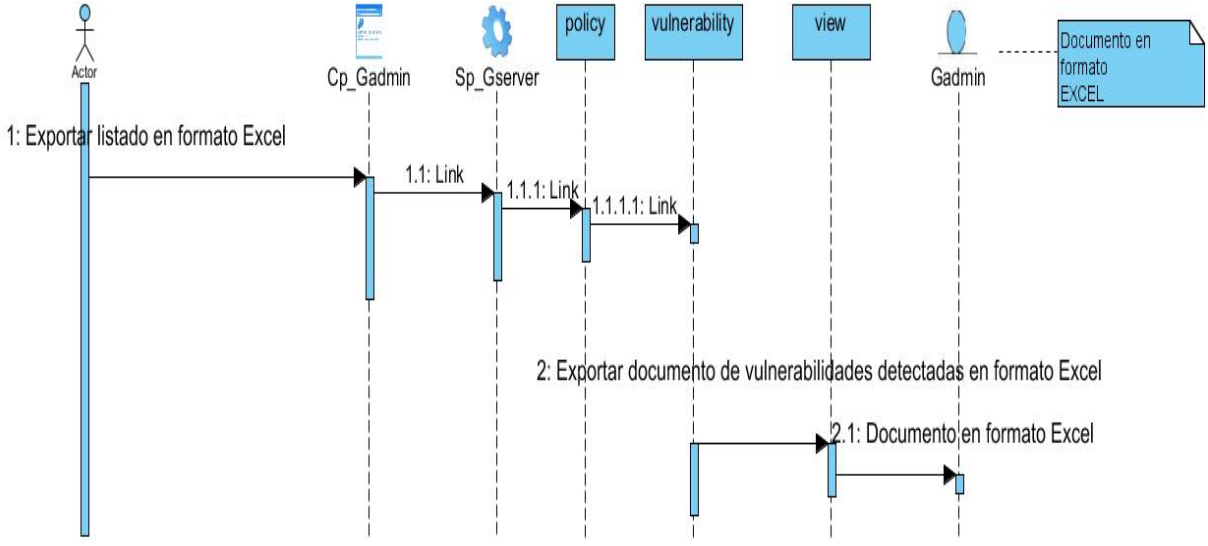

Figura 9: Diagrama de secuencia del Requisito Funcional (Exportar listado de vulnerabilidades detectadas en formato Excel.

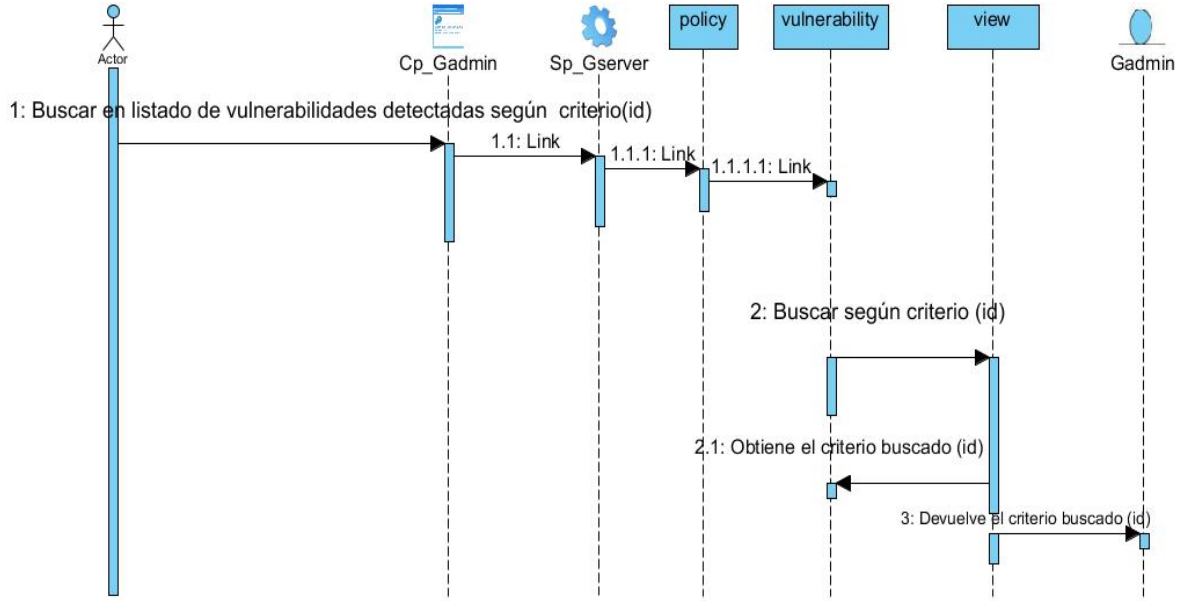

Figura 10: Diagrama de secuencia del Requisito Funcional (Buscar en el listado de vulnerabilidades detectadas según algún criterio)

2.10- Conclusiones del capítulo

En este capítulo se realizó el análisis y diseño de la propuesta de solución, para ello se elaboró un modelo conceptual, se identificaron los requisitos funcionales y no funcionales, se definieron los casos de usos, así como su diagrama y descripción. Además, se definió el modelo de datos, la arquitectura del sistema, el patrón arquitectónico y los diagramas de secuencia para dar solución a la investigación. Lo anteriormente planteado permitió una mejor comprensión del funcionamiento del negocio, posibilitando la implementación del módulo de detección de vulnerabilidades en la red mediante XILEMA GRHS.

Capítulo III. Implementación y prueba

El presente capítulo muestra los estándares de codificación definidos para el sistema, así como la implementación del diagrama de despliegue correspondiente al mismo. Además, se detallan las pruebas realizadas al sistema.

3.1- Modelo de implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema (Rivera, 2008).

Estándares de codificación

Se definen estándares de codificación porque un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible (Calleja, 2017). Con el fin de facilitar el entendimiento del código y fijar un modelo a seguir, se establecieron los siguientes estándares de codificación tomados del proyecto XILEMA GRHS:

- **CapWords**

Para nombrar las clases, definiendo que las mismas deben estar en el idioma inglés, y en caso de estar compuesta por más de una palabra debe utilizarse guion bajo entre las palabras. En la figura 11, se demuestra la utilización del estándar en el sistema.

```
@staticmethod
def class_name():
    return 'Vulnerability'

@staticmethod
def class_name():
    return 'Shared Folder'

@staticmethod
def class_name():
    return 'Screen Lock'
```

Figura 11: Ejemplo del uso del estándar CapWords

- **lower_case_with_underscores**

Para nombrar los métodos del programa principal, definiendo que las palabras son escritas en minúsculas separadas por guion bajo En la figura 12, se demuestra la utilización del estándar en el sistema.

```
    def get_queryset(self):

        nm = nmap.PortScanner()
        nm.command_line()
        nm.scan('127.0.0.1', '443')

        for host in nm.all_hosts():
            print('Host : %s (%s)' % (host, nm[host].hostname()))
            print('State : %s' % nm[host].state())
            for proto in nm[host].all_protocols():
                print('----------')
                print('Protocol : %s' % proto)
```

Figura 12: Ejemplo del uso del estándar lower_case_with_underscores

3.2- Diagrama de despliegue

Un modelo de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos. Un nodo es un elemento de hardware o software (Cillero, 2019). El diagrama de despliegue correspondiente para el módulo de detección de vulnerabilidades en la red para el sistema XILEMA GRHS es el siguiente:

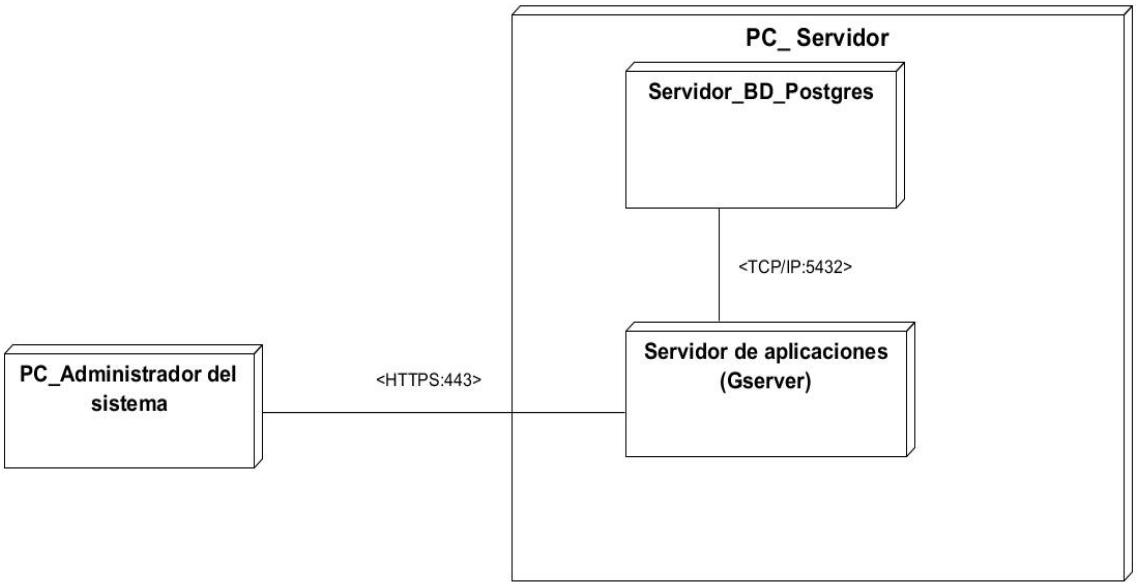

Figura 13: Diagrama de despliegue

Como se muestra en la figura 13, la distribución física del sistema en tiempo de ejecución consta de dos nodos. En uno se encuentra la PC_Administrador del sistema, que debe tener instalado un navegador web, el cual hace peticiones a la PC servidor donde se encuentra la aplicación XILEMA GRHS usando el protocolo de comunicación seguro HTTPS por el puerto 443. El sistema establece comunicación con el servidor de base dato PostrgresSQL utilizando el protocolo TCP/IP por el puerto 5432. De esta

forma se mantiene la seguridad en la PC servidor con la base de dato y con la aplicación. En la PC_Servidor se tiene la base de datos y la aplicación, pero de forma separada cada servicio es independiente, pero en la misma PC.

3.3- Prueba del software

La prueba de software es un elemento crítico para la garantía del correcto funcionamiento del software. Entre sus objetivos están: detectar defectos en el software, verificar la integración adecuada de los componentes y verificar que todos los requisitos se han implementado correctamente. Además de identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente y diseñar casos de prueba que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo (Pressman, 2010).

- **Estrategias de pruebas**

Las estrategias de pruebas de software proporcionan una guía que describe los pasos que deben realizarse como parte de la prueba, cuándo se planean y se llevan a cabo dichos pasos, y cuánto esfuerzo, tiempo y recursos se requerirán. Por tanto, cualquier estrategia de prueba debe incorporar la planificación, diseño, ejecución, recolección y evaluación de los casos de prueba (Pressman, 2010).

3.4- Disciplinas de prueba

La metodología AUP para la UCI plantea 3 disciplinas para guiar el desarrollo de las pruebas, que son: Pruebas internas, aceptación y liberación. A continuación, se describe la disciplina presente en la investigación:

Pruebas internas: En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como

intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de prueba ejecutables para automatizar las pruebas (Sánchez 2015).

3.5- Nivel de prueba utilizado

A continuación, se puntualiza el nivel de prueba correspondiente a la disciplina de pruebas descrita anteriormente. Dichos niveles propiciaron la detección de los errores existentes.

Unitarias: Las pruebas de unidad se concentran en la lógica del procesamiento interno y en las estructuras de datos de los límites de un componente. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Se puede decir que consiste en aislar cada parte del programa (clases, módulos, objetos, paquetes, subsistemas) para comprobar que cada una de esas partes funciona correctamente por separado (Jacobson, James y Booch, 2014).

3.6- Método de prueba

La metodología propone 2 métodos fundamentales que serán usados en el proceso de desarrollo de las pruebas al sistema, los cuales serán puntualizados a continuación:

- **Pruebas de caja negra**

Las pruebas de caja negra también denominadas pruebas de comportamiento, se centran en los requisitos funcionales del software. O sea, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (Pressman, 2010). Esta prueba

intenta encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructura de datos o en accesos a bases de datos externas, errores de rendimiento y errores de inicialización y terminación.

Se hace uso de la técnica partición equivalente, la que permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico.

- **Pruebas de caja blanca**

Las pruebas de caja blanca, denominadas a veces pruebas de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba (Pressman, 2010). Con la aplicación de este método se garantiza que se verifique por lo menos una vez todos los caminos independientes de cada módulo, que se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas, que se ejecuten todos los bucles en sus límites y con sus límites operacionales y se entrenen las estructuras internas de datos para asegurar su validez.

Se hace uso de la técnica del camino básico que permite derivar casos de prueba a partir de un conjunto de caminos independientes por los cuales puede circular el flujo de control.

3.7- Casos de prueba

Un caso de prueba es una especificación de un caso para probar el sistema, incluyendo qué probar, con qué entradas y resultados y bajo qué condiciones. Su principal objetivo es obtener un conjunto de pruebas que tengan una mayor

probabilidad de descubrir los defectos del software (ISTQB, 2011). Las entradas representan las variables que se pueden especificar y las mismas contienen: V, I. V indica válido e I indica inválido y N/A que no es necesario proporcionar un valor de la variable en un determinado caso, ya que es irrelevante.

En las siguientes tablas, se describieron los casos de pruebas para cada caso de usos implementados, haciendo uso del método de prueba caja negra y de la técnica de partición equivalente.

Tabla 18: Diseño de caso de prueba del caso de uso Exportar listado de vulnerabilidades detectadas en formato Excel

Escenario	**Descripción**	**Nombre**	**Respuesta del sistema**	**Flujo central**
EC 2.1 Exportar listado de vulnerabilidades detectadas en formato Excel	Se selecciona el botón exportar a Excel, seguidamente se genera el documento en ese formato.	*V*	El sistema muestra un mensaje informando que el documento fue exportado correctamente.	1-Autenticarse en el sistema. 2- Seleccionar la opción inventario. 3-Seleccionar la opción política. 4-Seleccionar la opción exportar a Excel.
EC 2.2 Seleccionar la opción cancelar.	El usuario selecciona la opción cancelar en el navegador cuando	NA	El sistema deshace y cancela la descarga.	1-Autenticarse en el sistema.

	se esté descargando el documento.			2- Seleccionar la opción inventario. 3-Seleccionar la opción política. 4-Seleccionar la opción exportar a Excel.

Tabla 19: Diseño de caso de prueba del caso de uso Buscar en el listado de vulnerabilidades detectadas según algún criterio

Escenario	**Descripción**	**Nombre**	**Respuesta del sistema**	**Flujo central**
EC 2.1 Buscar en el listado de vulnerabilidades detectadas según algún criterio	Se selecciona el cuadro de texto Filtrar búsqueda, seguidamente se abre un cuadro de texto para que el usuario busque según los criterios establecidos.	*V*	El sistema muestra un mensaje informando que fue encontrado con suceso.	1-Autenticarse en el sistema. 2- Seleccionar la opción inventario. 3-Seleccionar la opción política. 4-Seleccionar la opción Filtrar búsqueda.
EC 2.2 Valor incorrecto	El usuario selecciona la opción filtra la búsqueda y escribe	I	El sistema emite un mensaje de que el criterio	1-Autenticarse en el sistema.

	un valor incorrecto, por ejemplo: letras para filtrar IP, Fecha en el formato incorrecto.		seleccionado es incorrecto.	2- Seleccionar la opción inventario. 3-Seleccionar la opción política. 4-Seleccionar la opción Filtrar búsqueda.

3.8- Resultado de las pruebas

- **Prueba de caja blanca**

Para desarrollar las pruebas de caja blanca, se ha utilizado la técnica del camino básico, en seguida se detalla cómo funciona dicha técnica.

Pasos para el desarrollo de la técnica del Camino básico:

1. Se escoge el diseño o el código de la aplicación como base, y se dibuja el correspondiente grafo de flujo. Se obtiene la complejidad ciclomática del grafo de flujo.

2. Se determina la complejidad ciclomática del grafo resultante, posteriormente se determinan los casos de pruebas que permitan la ejecución de los caminos anteriores.

3. Se establece un conjunto básico de caminos linealmente independientes.

4. Se elaboran los casos de pruebas que permitirán la ejecución de cada camino del conjunto básico.

Notación del grafo de flujo:

- **Nodo del grafo de flujo(N):** Es un círculo que representa las sentencias procedimentales.

- **Aristas(A):** Son las flechas o enlaces que representan el flujo de control del grafo.

- **Regiones:** Son las áreas delimitadas por las aristas y nodos. Al realizar el conteo de las regiones también se incluye el área exterior del grafo.

- **Complejidad ciclomática:** Se calcula a partir del grafo de control del software bajo análisis y mide, a grandes rasgos, el número de caminos independientes que pueden ocurrir en la ejecución, entre el inicio y el fin del software.

  Se define como: V (G)= A-N+2.

  A continuación, se muestra un fragmento de código perteneciente al algoritmo desarrollado:

```
for host in nm.all_hosts():                                          1
    print('Host : %s (%s)' % (host, nm[host].hostname()))            2
    print('State : %s' % nm[host].state())
    if nm[host].state() == 'up':                                     3
        for proto in nm[host].all_protocols():                       4
            print('----------')
            print('Protocol : %s' % proto)                           5
        lport = nm[host][proto].keys()
        lport.sort()
        for port in lport:                                           6
            print('port : %s\tstate : %s' % (port, nm[host][proto][port]['state']))
            print('127.0.0.1', '21-1024')                           7
    else:
        print('No hay computadoras encendidas')                      8
```

Figura 14: Fragmento de código para prueba.

Posteriormente se procede a la elaboración del grafo de flujo teniendo en cuenta dicha enumeración, como se observa en la figura 15.

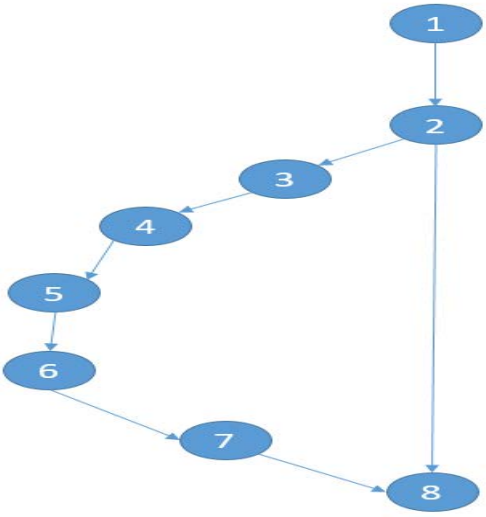

Figura 15: Representación del grafo de flujo de camino básico

Complejidad ciclomática

V (G)= A-N+2

8-8+2=2

Caminos independientes

1,2,8

1,2,3,4,5,6,7,8

Cada camino independiente es un caso de prueba a realizar, de forma que se garantiza que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. En el caso anterior se calcularon 2 caminos básicos, por tanto, es igual al número de casos de prueba realizados.

Tabla 20: Caso de prueba de caja blanca para el camino básico 1.

Entrada	***N/A***
Resultados Esperados	*No se pudo obtener el estado de la computadora (encendido, apagado).*
Condiciones	*El sistema emite un mensaje de error.*

Tabla 21: Caso de prueba de caja blanca para el camino básico 2.

Entrada	***N/A***
Resultados Esperados	*Se obtiene el estado de los puertos escaneados.*
Condiciones	*Se imprime un mensaje del estado del puerto (abierto, cerrado, filtrado)*

Una vez realizadas las pruebas unitarias mediante la técnica de camino básico al código seleccionado, se puede afirmar que la prueba concluye de forma satisfactoria, por tanto, se obtuvo el resultado esperado.

- **Aplicación de prueba de caja negra (casos de prueba)**

Apoyados en el diseño de casos de prueba se realizaron 2 iteraciones de pruebas internas pertenecientes al nivel de sistema. A continuación, se presentan los resultados arrojados durante las diferentes pruebas aplicadas:

Tabla 22: Resultados de la prueba de caja de negra 1ra iteración

Casos de Prueba	***No Conformidades***			
	Alta	*Media*	*Baja*	*Total*
Exportar documento en el formato Excel	*0*	*2*	*0*	*2*
Buscar en el listado de vulnerabilidades detectadas según algún criterio	*1*	*2*	*3*	*6*
Cantidad de memoria RAM	*0*	*2*	*2*	*4*
Cantidad de CPU	*2*	*2*	*4*	*8*
Total.	*3*	*8*	*9*	*20*

Tabla 23: Resultados de la prueba de caja de negra 2a iteración

Casos de Prueba	***No Conformidades***			
	Alta	*Media*	*Baja*	*Resuelta*
Exportar documento en el formato Excel	*0*	*2*	*0*	*2*
Buscar en el listado de vulnerabilidades detectadas según algún criterio	*0*	*2*	*3*	*5*
Cantidad de memoria RAM	*0*	*0*	*0*	*0*
Cantidad de CPU	*1*	*0*	*1*	*2*
Total.	*1*	*4*	*4*	*9*

La realización de pruebas (caja negra y caja blanca) al módulo permitió detectar varias no conformidades en las primeras iteraciones siendo estas resueltas. Las no conformidades se clasificaron en Alta, Media o Baja en dependencia del impacto que tuvieran. A continuación, en la figura 16, se muestra la relación de no conformidades (detectadas y resueltas) por iteración:

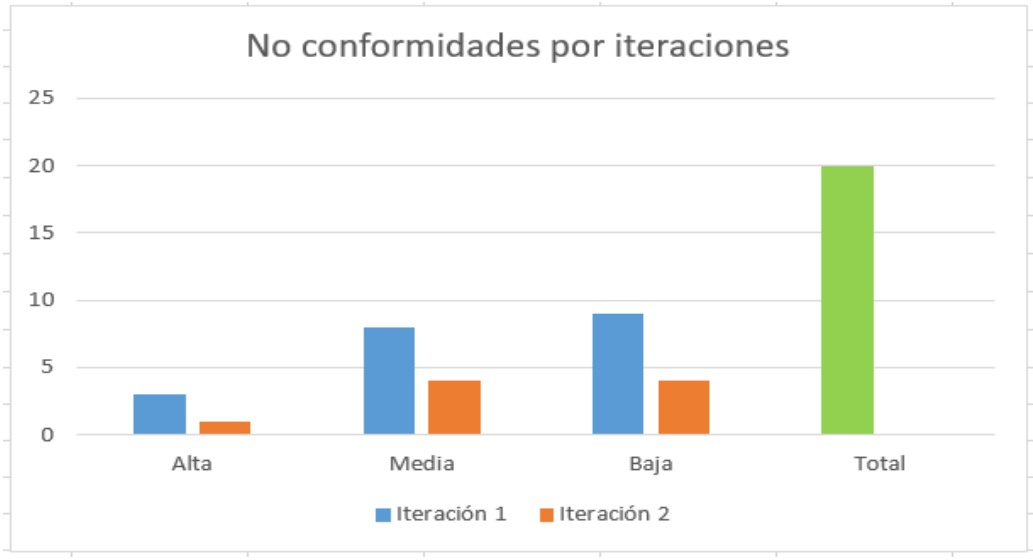

Figura 16: Resultado de las pruebas

Como se puede apreciar en la figura se realizaron dos iteraciones de pruebas. A lo largo de la primera iteración se detectaron 20 no conformidades, asociadas a errores de interfaz y a errores ortográficos, todas fueron resueltas en la propia iteración. En una segunda y última iteración no se detectó ninguna no conformidad por lo que la aplicación mostró un buen funcionamiento y se considera terminada.

3.9- Conclusiones parciales

Al realizar la implementación del sistema siguiendo los estándares de codificación definidos, se desarrolló un código reutilizable y comprensible por los integrantes del equipo de desarrollo. Se realizó la descripción de la implementación del componente para el módulo a través del diagrama de despliegue, el cual facilitó mostrar la disposición de las particiones físicas del sistema y la asignación de los componentes de software a estas particiones. Se definieron las pruebas a aplicar para comprobar el correcto funcionamiento de la aplicación.

Conclusiones generales

Durante el desarrollo de la investigación se planteó la necesidad de diseñar e implementar un módulo para contribuir en el proceso de detección de detección de puertos abiertos y las vulnerabilidades que representan en la red para integrar al sistema XILEMA GRHS, dándole así cumplimiento al objetivo propuesto de la siguiente forma:

- La elaboración del marco teórico conceptual de la investigación permitió la realización de un marco de referencia para conformar una solución bien fundamentada, aportando ideas para el desarrollo del módulo para el sistema XILEMA GRHS.
- A partir de la realización del análisis y diseño del módulo se obtuvo como resultado los diagramas y artefactos necesarios para guiar el desarrollo del mismo.
- El desarrollo de las pruebas de caja negra y caja blanca permitió detectar y corregir los errores detectados, posibilitando cumplir con las especificaciones requeridas y la validación del sistema implementado.

Recomendaciones

En función del constante proceso de mejora y evolución que es inherente a todo sistema de software se recomienda lo siguiente:

- Integrar el módulo al sistema XILEMA GRHS.

- Integrar la posibilidad de cuando el sistema detecte las vulnerabilidades generadas por los puertos se puedan tomar acciones de control (cerrar los puertos abiertos).

- Realización de pruebas funcionales y no funcionales.

Referencias Bibliográficas

1. ALLAMANDRI, M., 2012. Modelo de diseño de software by Maxi Allamandri on Prezi. [en línea]. [Consulta: 28 febrero 2020]. Disponible en: https://prezi.com/p4vuh3fapd5c/modelo-de-diseno-de-software/.

2. BEALE, J., MEER, H., WALT, C. van der y DERAISON, R., 2016. *Nessus Network Auditing: Jay Beale Open Source Security Series*. S.l.: Elsevier. ISBN 978-0-08-047962-0.

3. CALLEJA, M.A., 2017. Estándares de codificación. *Carmen. Estándares de codificación*. S.l.: s.n., pp. 1-9.

4. CASTELLS, M., 2013. El impacto de internet en la sociedad: una perspectiva global. [en línea]. University of Southern California, Los Angeleles, E.E.U.U, 2013. Disponible en: https://www.bbvaopenmind.com/articulos/el-impacto-de-internet-en-la-sociedad-una-perspectiva-global/.

5. CEVALLOS, K., 2015. UML: Diagrama de secuencia. [en línea]. Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López. Disponible en: https://ingsoftwarekarlacevallos.wordpress.com/2015/07/07/uml-diagrama-de-secuencia/.

6. CILLERO, M., 2019. Diagrama de Despliegue. *manuel.cillero.es* [en línea]. [Consulta: 23 febrero 2020]. Disponible en: https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-despliegue/.

7. CYTA, 2018. *Entorno de Desarrollo Integrado (IDE).* [en línea]. S.l.: s.n. Disponible en: http://www.cyta.com.ar/ta1401/v14n1a2.htm.

8. DJANGO, 2015. Django documentation | Django documentation | Django. [en línea]. [Consulta: 28 febrero 2020]. Disponible en: https://docs.djangoproject.com/en/2.0/.

9. FERNÁNDEZ, L.F., 2006. Arquitectura de software. *Software Guru*, vol. 2, no. 3, pp. 40–45.

10. GARCÍA, S., 2015. *La guía definitiva de Django: Desarrolla aplicaciones web de forma rápida y sencilla* [en línea]. S.l.: s.n. Disponible en: http://github.com/saulgm/djangobook.com.

11. GARCÍA-PEÑALVO, F.J., 2019. Procesos y Métodos de Modelado para la Ingeniería Web. ,

12. GONZÁLEZ, C.J. y RODRÍGUEZ, R.V., 2017. *Editor web visual para HTML, CSS y JavaScript de apoyo a la docencia* [en línea]. S.l.: s.n. 14. Disponible en: http://librosweb.es/libro/javascript/.

13. GUZMÁN, R. y RODRÍGUEZ, S., 2017. *Sistema para la detección automática de publicación de servicios telemáticos no autorizados en la red de la Universidad de las Ciencias Informática*s. Trabajo de diploma. La Habana, Cuba: Universidad de las Ciencias Informáticas.

14. INFANTE, C.A.F., 2013. *GUÍA DE ELABORACIÓN DE MODELOS*. Bogotá D.C, Colombia: s.n.

15. INGENIODS, 2013. Patrones Arquitectónicos. *Ingenio DS* [en línea]. [Consulta: 28 febrero 2020]. Disponible en: https://ingeniods.wordpress.com/2013/09/16/patrones-arquitectonicos/.

16. JACOBSON, I., JAMES, R. y BOOCH, G., 2014. *El Proceso Unificado de Desarrollo de Software*. S.l.: s.n.

17. JAVIER, A.G., 2015. *Derecho penal y redes sociales* | [en línea]. Pamplona, España: Aranzadi. [Consulta: 2 diciembre 2019]. Disponible en: http://www.marcialpons.es/libros/derecho-penal-y-redes-sociales/9788490983263/.

18. JETBRAINS, 2017. PyCharm: the Python IDE for Professional Developers by JetBrains. *JetBrains* [en línea]. [Consulta: 2 diciembre 2019]. Disponible en: https://www.jetbrains.com/pycharm/.

19. JULIÁN PÉREZ PORTO y MARÍA MERINO, 2016. Definición de lenguaje de programación - Qué es, Significado y Concepto. [en línea]. [Consulta: 14 febrero 2020]. Disponible en: https://definicion.de/lenguaje-de-programacion/.

20. LARMAN, Craig, 2003. Diseño de las realizaciones de casos de uso con los patrones de diseño GoF. *UML y patrones*. Segunda edición. Pretince-Hall: s.n., pp. Capítulo 23. ISBN 84-205-3438-2.

21. LUJÁN-MORA, S. y VALAREZO, E., 2014. Aplicaciones Web - Patrones de diseño. [en línea], [Consulta: 28 febrero 2020]. Disponible en: http://rua.ua.es/dspace/handle/10045/36735.

22. LYON, G., 2017. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. S.l.: Insecure.

23. MAIDA, E.G. y PACIENZIA, J., 2015. *Metodologías de desarrollo de Software* [en línea]. Tesis de Licenciatura. Buenos Aires: Pontificia Universidad Católica Argentina. Disponible en:

https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf.

24. MARÍN, R., 2019. Los gestores de bases de datos (SGBD) más usados. *Canal Informática y TICS* [en línea]. [Consulta: 2 diciembre 2019]. Disponible en: https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/.

25. Metodologías de desarrollo de Software - EcuRed. [en línea], [sin fecha]. [Consulta: 23 enero 2019]. Disponible en: https://www.ecured.cu/Metodologias_de_desarrollo_de_Software.

26. MORAN, N. y ARAUJO, A., 2018. Patrones de diseño en el framework Django. *Web frameworks y patrones de diseño* [en línea]. Ingeniería Universidad de Los Andes Mérida, Venezuela: s.n., pp. 13-22. [Consulta: 5 junio 2020]. Disponible en: https://b1b10bmlvabco.couldfront.net/attach/jbqz98tozml4ia/iexj3a76c1s2oi/jfou8ccf0w7ny/frameworks.pdf.

27. POSTGRESQL, G.D.G., 2018. PostgreSQL: About. [en línea]. [Consulta: 2 diciembre 2019]. Disponible en: https://www.postgresql.org/about/.

28. PRESSMAN, R., 2005. Ingeniería de Software Un Enfoque Práctico .6th.edicion-. *Scribd* [en línea]. [Consulta: 28 febrero 2020]. Disponible en: https://es.scribd.com/doc/27182020/Ingenieria-de-Software-Un-Enfoque-Practico-6th-edicion-Roger-pressman-1.

29. PRESSMAN, Roger S., 2002. *Ingeniería de Software, un enfoque práctico.* Quinta. S.l.: McGraw-Hill Companies.

30. PRESSMAN, R.S., 2010. *Ingeniería de software: Un enfoque práctico*. Septima Edición. S.l.: s.n.

31. PYPI.ORG, 2016. *Python-Nmap* [en línea]. 29 junio 2016. S.l.: s.n. Disponible en: https://pypi.org/project/python-nmap/.

32. PYPI.ORG, 2019. *python-masscan* [en línea]. 29 junio 2019. S.l.: s.n. Disponible en: pypi.org/project/python-masscan/.

33. PYTHON.ORG, 2019. Welcome to Python.org. [en línea], Disponible en: https://www.python.org/.

34. QUEZADA, A.C., 2017. *Escaneo de Puertos utilizando Masscan* [en línea]. S.l.: s.n. Disponible en: http://www.reydes.com/d/?q=Escaneo_de_Puertos_utilizando_Masscan.

35. RIVERA, E., 2008. *Arquitectura de software II- Diagrama de Componentes y Despliegue* [en línea]. S.l.: s.n. Disponible en: https://es.scribd.com/document/7884665/Arquitectura-de-Software-II-Diagrama-de-Componentes-y-Despliegue.

36. SAAVEDRA, J., 2007. Patrones Grasp | El Mundo Informático. [en línea]. [Consulta: 28 febrero 2020]. Disponible en: https://jorgesaavedra.wordpress.com/category/patrones-grasp/.

37. SÁNCHEZ, T.R., 2015. 4.2 Descripción de las disciplinas. *Metodología de desarrollo para la Actividad productiva de la UCI*. Universidad de las Ciencias Informáticas Carretera a San Antonio Km 2 1⁄2. Torrens. Boyeros. La Habana. Cuba: s.n., 1.2,

38. SÁNCHEZ, T.R., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.* 1.2. S.l.: s.n.

39. SOMMERVILLE, I., 2006. Software Engeneering. *Software Engeneering 07 Edition*. United Kingdom: s.n., pp. 110.

40. SOMMERVILLE, I., 2006. Software Engeneering. *Software Engeneering 07 Edition*. United Kingdom: s.n., pp. 111.

41. TARAZONA, I. y GOMÉZ, O., 2016. *Uso del UML en el Modelado de Datos* [en línea]. S.l.: s.n. Disponible en: https://es.scribd.com/document/8962964/uml.

42. TENABLE, 2015. Nessus. *Tenable*® [en línea]. [Consulta: 8 marzo 2020]. Disponible en: https://es-la.tenable.com/products/Nessus/Nessus-professional.

43. UNAD, 2020. Diagramas de Casos de Uso | LENGUAJE DE MODELADO UNIFICADO UML. [en línea]. [Consulta: 17 febrero 2020]. Disponible en: http://stadium.unad.edu.co/ovas/10596_9839/diagramas_de_casos_de_uso.html.

Glosario de términos

AUP: *Agile Unified Process* (Proceso Unificado Ágil en español) es una metodología de desarrollo de software de tipo ágil.

AUP-UCI: Proceso Unificado Ágil para el Desarrollo de la Actividad Productiva en la Universidad de las Ciencias Informáticas.

CASE: (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadoras): Son diversas Aplicaciones informáticas destinadas a aumentar la productividad en el Desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

CMMI (Integración de sistemas modelos de madurez de capacidades en español): Es un modelo para mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

Cliente XILEMA GRHS: Responsable de realizar el inventario de los componentes de Hardware y Software,

Exploits: Fragmento de datos o secuencia de comandos o acciones, utilizada con el fin de aprovechar una vulnerabilidad de seguridad de un sistema de información para conseguir un comportamiento no deseado del mismo.

Escáner de vulnerabilidades informáticas: Es un software que, dado un objetivo, ya sea una computadora o una red de computadoras, tiene como objetivo detectar los potenciales riesgos al que están expuestos los equipos seleccionados.

Firewall: Es un componente de un sistema informático que está diseñada para bloquear el acceso no autorizado.

Framework (Marco de Trabajo): Es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática que sirve de referencia, para enfrentar y resolver nuevos problemas de índole similar.

FTP (Protocolo de Transferencia de Archivos en español): Es un protocolo de rede de transferencia de archivos entre sistemas conectados entre sistemas conectados a red TCP.

HTML (Lenguajes de marcas de hipertexto en español): Es un lenguaje de marcado que se utiliza para el desarrollo de páginas de internet.

Host: El término host o anfitrión se usa en informática para referirse a las computadoras u otros dispositivos (tablets, móviles, portátiles...) conectados a una red que proveen y utilizan servicios de ella.

HTTP (Protocolo de transferencia de hipertexto): Es un protocolo de comunicación que permite las transferencias de información que permite las transferencias de información en la web.

HTTPRequest: Es un paquete/objeto de información que una computadora envía a otra computadora para comunicar algo.

IP: Es la sigla de *Internet Protocol* o, (Protocolo de Internet en español). Se trata de un estándar que se emplea para el envío y recepción de información mediante una red.

RUP: Proceso Unificado de Desarrollo (*por sus siglas en inglés Rational Unified Process*) es una metodología de desarrollo de software que está en componentes e interfaces bien definidas, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Políticas de seguridad informática: Son reglas que usan las organizaciones para salvaguardar la información de la empresa, para ello debe definir claramente sus objetivos de seguridad y así crear reglas y procedimientos que cada usuario debe seguir.

Red de computadoras: Una red de comunicaciones de datos o una red de computadoras es la interconexión de distinto número de sistemas informáticos a través de una serie de dispositivos de telecomunicaciones y un medio físico.

SMB (Bloque de mensajes del servidor en español): Es un protocolo de red que permite compartir archivos, impresoras, entre nodos de una red de computadoras que usan el sistema operativo Microsoft Windows.

Servidor: Un ordenador o software que ofrece servicios a máquinas de cliente distantes o a aplicaciones, como el suministro de contenidos de páginas (textos u otros recursos) o el retorno de los resultados de consultas.

TIC: Tecnologías de la Información y las Comunicaciones.

TCP (Protocolo de control de transmisión en español): Un sistema de protocolos que hacen posibles servicios Telnet, FTP, E-mail, y otros entre ordenadores que no pertenecen a la misma red.

Vulnerabilidad en el software: Es un fallo o debilidad en el diseño, la implementación, el funcionamiento, o la gestión de un sistema, que puede ser explotado con la finalidad de violar la política de seguridad del sistema.

Anexo 1

La entrevista se realizó de manera informal. Las preguntas fluyeron en dependencia del conocimiento del cliente sobre el modulo implementado.

1. ¿Qué requisitos debe tener el módulo de detección de vulnerabilidades en la red para el sistema XILEMA GRHS?

2. ¿Cuál es el objetivo de implementar este módulo?

3. ¿Qué datos se manejan en el módulo?

4. ¿Las vulnerabilidades detectadas, son para tomar acciones?

5. ¿Quiénes deben tener acceso al módulo?

6. ¿Cómo garantizar la seguridad del módulo?

Especialista A del centro TLM: Ramón Guzmán Alemañy